INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Beurteilung der Verwundbarkeit des Globus Toolkit 5 mit Hilfe der First Principles Vulnerability Assessment-Methode

Michael Goldberger

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Beurteilung der Verwundbarkeit vom Globus Toolkit 5 mit Hilfe der First Principles Vulnerability Assessment-Methode

Michael Goldberger

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Dr. Michael Schiffers
Abgabetermin:	31. März 2011

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 28. März 2011

.....
(Unterschrift des Kandidaten)

Abstract

In dieser Arbeit wird eine Komponente des Globus Toolkit 5, einer gängigen Grid Middleware, auf mögliche Schwachstellen hin untersucht und hinsichtlich ihrer Verwundbarkeit bewertet. Gegenstand der Untersuchung ist das Grid Resource Allocation and Management Interface in der aktuellen Version 5. Die Suche erfolgt mit Hilfe der First Priciples Vulnerability Assessment Methode, einem von den Universiäten von Wisconsin und Barcelona kürzlich vorgestellten Ansatz zur manuellen Analyse auf Basis vorhandener Quelltexte. Die Ergebnisse der Untersuchung bilden, zusammen mit den bei der Durchführung der Analyse gemachten Erfahrungen, die Basis für eine Bewertung der angewandten Methode. Hauptkriterium ist hierbei, neben einer ausreichenden Erkennungsrate von Schwachstellen, die Fähigkeit den Aufwand, der bei einer Untersuchung von komplexen, verteilten Anwendungen betrieben werden muss, zu reduzieren.

Inhaltsverzeichnis

1 Einführung und Motivation

1.1 Fragestellung und Lösungsansatz

Unter dem Begriff Grid Computing versteht man die koordinierte Nutzung von Ressourcen wie Rechner, Daten und Software in dynamischen, multi-institutionalen Virtuellen Organisationen (VO) zur Lösung wissenschaftlicher Problemstellungen [1]. Realisiert werden Grid Systeme mit Hilfe von Grid Middleware, wie zum Beispiel dem Globus Toolkit, UNICORE oder glite, welche auf Basis von offenen und standardisierten Protokollen und Schnittstellen Komponenten bereitstellen mit denen komplexe Grid Infrastrukturen aufgebaut werden können. Genutzt werden sie meist für umfangreiche Berechnungen die viele Ressourcen benötigen und eine lange Laufzeit aufweisen. Beispiele sind Simulationen von subatomaren Abläufen, sowie die Berechnung von Ergebnissen im Zusammenhang mit an Teilchenbeschleunigern gewonnenen Daten an der Europäischen Organisation für Kernforschung (CERN) [2]. Auch lösen sich in letzter Zeit Grid Infrastrukturen von der ausschließlich wissenschaftlichen Verwendung und halten in der Wirtschaft unter dem Namen „enterprise grid computing“ Einzug. Anwendungsgebiete können hier die Berechnung von wirtschaftlichen Vorhersagen oder die Simulation von Crash-Test sein.

Zur Erhöhung der Akzeptanz von Grid Infrastrukturen in der Wirtschaft ist es unerlässlich, dass Grid Systeme zuverlässig arbeiten, da die auszuführenden Berechnungen in der Regel eine lange Laufzeit aufweisen und oft sensible Daten genutzt werden. Will man nun eine fundierte Aussage bezüglich der Zuverlässigkeit eines Grid Systems treffen, ist es jedoch nötig die Verwundbarkeit der zugrunde liegenden Middleware zu beurteilen. Einen potentiellen Angreifer darf es nicht möglich sein, durch das Ausnutzen von Schwachstellen die Verfügbarkeit der Ressourcen zu verhindern, vertrauliche Daten zu erlangen oder zu verändern sowie die Ressourcen unautorisiert zu nutzen. Ein Grid System ist aufgrund der Dynamik der Virtuellen Organisationen mehr Bedrohungen ausgesetzt als ein konventionelles vernetztes System. Die Dynamik drückt sich dabei im Äußeren durch ihren Lebenszyklus und im Inneren durch den häufigen Wechsel von Ressourcen, Diensten und Mitglieder aus. Damit erhöhen sich die Möglichkeiten, die einem eventuellen Angreifer zu Verfügung stehen. Zwar gibt es eine Reihe von Arbeiten, die sich mit den einzelnen Aspekten dieses Themas beschäftigen, aber die systematische Beurteilung der Verwundbarkeit (vulnerability assesment) wird in den meisten Grid- und Cloud-Projekten nur unzureichend behandelt. Ein systematisches Vorgehen ist dabei unerlässlich, da aufgrund der Komplexität von Grid Systemen und der damit verbundenen hohen Anzahl von Codezeilen, eine zeilenweise Untersuchung des kompletten Quellcodes auf mögliche Schwachstellen jeglichen Rahmen sprengen würde. Die angewandte Methodik muss für eine erfolgreiche Untersuchung die Arbeitsweise und die besonderen Problematiken der untersuchten Systeme berücksichtigen können. Im Fall von Grid Systemen also die Verteiltheit der Systemkomponenten sowie die Dynamik der Ressourcen.

Als ein möglicher Ansatz um Verwundbarkeitsanalysen von Middleware in verteilten Systemen systematisch durchzuführen wurde kürzlich von den Universitäten von Barcelona und Wisconsin die von ihnen entwickelte First Principles Vulnerability Assessment Methode (FPVA) [3] vorgeschlagen. Die Methode gehört der Gruppe der sogenannten White-Box Verfahren an, welche genutzt werden, um statische Codeanalysen auf Basis von vorhandenen Quelltexten durchführen. Durch einen klar definierten Prozess versucht sie den Aufwand zu minimieren, der betrieben werden muss um eventuell vorhandene Schwachstellen aufzuspüren.

Von der Gruppe der Universität von Wisconsin wurde in der Vergangenheit bereits mit der Einführung des Fuzz Testing, ein Black-Box Verfahren zur Prüfung der Zuverlässigkeit von Software mittels Zufallsdaten, ein in vielen Projekten angewandtes dynamisches Verfahren zur Analyse der Sicherheit und Zuverlässigkeit entwickelt [4]. Black-Box Verfahren, also Verfahren auf Basis kompilierter Programme ohne Zugang zu den Quelltexten, eignen sich primär dazu das Vorhandensein von Schwachstellen zu beweisen. Sie können diese aber in der Regel nicht in den Quelltexten lokalisieren. White-Box Verfahren ermöglichen es hingegen, gezielt nach den möglichen Schwächen im Code zu suchen um diese zu schließen. Obwohl beide Verfahren nur die Existenz von Schwachstellen nachweisen können, nicht jedoch deren Abwesenheit, sind sie in der Lage die Sicherheit eines System erheblich zu erhöhen. Auch eignen sie sich dazu unterschiedliche Systeme bezüglich ihrer Angreifbarkeit, und damit ihrer Zuverlässigkeit, zu vergleichen.

Es ist nun zu prüfen, ob die First Principles Vulnerability Assessment Methode den Anforderungen einer systematischen Beurteilung der Verwundbarkeit von Middleware im Grid Umfeld entspricht, also ob sie in der Praxis eingesetzt werden kann, um die einzelnen Komponenten von Grid Middleware Systeme gezielt nach Schwachstellen zu untersuchen und diese dadurch, hinsichtlich ihrer Verwundbarkeit und ihrer Zuverlässigkeit, miteinander zu vergleichen. Dabei muss die Methode die Möglichkeit bieten, den erforderlichen Aufwand zu minimieren und gleichzeitig eine hohe Erkennungsrate aufweisen.

Um eine fundierte Aussage treffen zu können, soll dies anhand einer Fallstudie exemplarisch getestet werden. Dazu wird im Verlauf dieser Arbeit auszugsweise eine Komponente des Globus Toolkits in der aktuellen Version 5 (GT5), eine gängige Grid Middleware, mit Hilfe der First Principles Vulnerability Assessment-Methode bewertet. Die Ergebnisse sowie die Erfahrungen bei der Anwendung werden zur Bewertung der vorgestellten Assessment Methode herangezogen. Auch können Teile dieser Arbeit als Schritt für Schritt Anleitung gesehen werden, um die First Principles Vulnerability Assessment Methode auf andere Systeme anzuwenden. Die Wahl der untersuchten Middleware fiel auf das Globus Toolkit 5, da dieses im Vergleich zu seinem Vorgänger, dem Globus Toolkit 4, eine auffällige Wandlung im Bereich der Ressourcen Zuteilung und Verwaltung erfahren hat. Die gewonnenen Ergebnisse werden auch dazu verwendet eine grundsätzliche Aussage über die Verwundbarkeit der untersuchten Komponente, dem Grid Resource Allocation and Management 5 (GRAM5), zu geben.

1.2 Vorgehensweise

Die vorliegende Arbeit gliedert sich, einschließlich dieser Einleitung, in 5 Kapitel, die folgende Themen behandeln:

Kapitel 2 (Begriffliche Einordnung und Grundlagen) bildet das begriffliche Grundgerüst, indem es den Begriff der Verwundbarkeit diskutiert, die Vorgehensweise der First Principles Vulnerability Assessment Methode vorstellt und die dem Globus Toolkit zugrunde liegende Architektur und die von ihm bereitgestellten Komponenten erörtern.

Kapitel 3 (Verwandte Arbeiten) geht auf existierende Arbeiten im Bereich der Verwundbarkeitsanalyse mit der First Principles Vulnerability Assessment Methode ein, stellt eine bereits etablierte Methode im Bereich der Verwundbarkeitsanalyse vor und gibt einen kurzen Überblick über Programme zur automatischen Schwachstellenanalyse.

Kapitel 4 (First Principles Vulnerability Assessment angewandt auf GRAM5) stellt die eigentliche Analyse des Grid Resource Allocation and Management 5 mit der First Principles Assessment Methode dar und arbeitet dabei die in Kapitel 2 vorgestellten Schritte ab. Auch wird hier auf Basis der Ergebnisse die Methode selbst diskutiert.

Kapitel 5 (Zusammenfassung und weiterführende Arbeiten) fasst die einzelnen Kapitel zusammen und beinhaltet ein Fazit sowie Hinweise auf mögliche Folgearbeiten, um die die vorliegende Arbeit erweitert werden kann.

2 Begriffliche Einordnung und Grundlagen

Dieses Kapitel schafft die Grundlagen für die in Kapitel 5 durchgeführte Verwundbarkeitsanalyse. Es wird der Begriff der Verwundbarkeit diskutiert und die Arbeitsschritte der First Principles Vulnerability Assessment Methode vorgestellt. Abschließend werden die Komponenten des Globus Toolkit 5 erörtert und die wesentlichen Unterschiede zu seinem Vorgänger, dem Globus Toolkit 4, aufgezeigt.

2.1 Verwundbarkeit

Im Zusammenhang mit Softwaresystemen versteht man unter dem Begriff der Verwundbarkeit die Existenz von Sicherheitslücken oder Schwachstellen (Vulnerabilities) in einer Software (Application), die von potentiellen Angreifern mittels Schadprogrammen (Exploits) ausgenützt werden können. Dies geschieht, um den Besitzern der Software, deren Benutzern oder anderen in Zusammenhang mit der Anwendung stehenden, Schaden zuzufügen bzw. um sich einen illegalen Vorteil zu verschaffen. Sie können durch Schwächen im Design der Software oder durch Fehler bei der Implementierung entstehen [5, 6].

Ein Großteil der existierenden Schwachstellen basieren auf eine unzureichende oder fehlerhafte Überprüfung der Eingabedaten. Oft reicht es, wenn sich der Programmierer bei der Größenangabe von Speicherbereichen um eine Stelle verrechnet. Bei Buffer Overflow Schwachstellen kann dann beispielsweise über den reservierten Speicherbereich hinaus geschrieben werden, was zu unvorhersehbaren Reaktionen, bis hin zum Absturz des Programms, führen kann. Erfolgt ein gezielter Angriff (Attack) auf die Schwachstelle, kann unter Umständen die Ausführung von beliebigen Code ermöglicht werden. Dies ist besonders kritisch, wenn die Anwendung mit privilegierten Rechten läuft, da dadurch eine vertikale Rechteausweitung erreicht werden kann. Andere Beispiele von Schwachstellen die durch eine unzureichende Überprüfung der Eingabedaten entstehen, sind SQL-Injection und Cross Site Scripting Schwachstellen. Weitere Sicherheitslücken existieren, wenn Aktionen nicht ausreichend protokolliert werden oder die Möglichkeit besteht die geloggten Daten zu verändern, wenn Passwörter fest in den Quellcode eingebunden oder schwache Passwörter erlaubt sind.

Es existieren eine Vielzahl von bekannten Schwachstellen die, wenn sie von gutmütigen Institutionen entdeckt und gemeldet wurden, von verschiedenen Projekten veröffentlicht werden. Beispiele sind die unterschiedlichen Computer Emergency Response Teams (CERT) [7] oder das Open Web Application Security Project (OWASP) [5]. Dass diese Listen keineswegs vollständig sind, zeigt das Aufkommen sogenannter Zero-Day-Vulnerabilities, also neu entdeckter Schwachstellen, für die auch ein florierender und finanziell attraktiver Schwarzmarkt existiert.

Schwachstellen werden oft fälschlicherweise mit den eigentlichen Angriffen gleichgesetzt. Bei einer eingehenden Untersuchung ist es aber wichtig zwischen den Schwachstellen, welche die Angriffsfläche bieten, den Angriffen, die diese Schwachstellen ausnutzen, und den eingesetzten Schadprogrammen, durch die Angriffe ausgeführt werden, zu unterscheiden.

In komplexen Softwaresystemen ist es praktisch nicht möglich Schwachstellen auszuschließen. Um dennoch eine höchst mögliche Sicherheit zu gewährleisten empfiehlt es sich, schon in der Entwicklung einige Dinge zu beachten. Beim Design sollten mögliche Abuse-Cases, also Möglichkeiten die Software anders als angedacht zu nutzen, betrachtet werden um diese auszuschließen. Auf als unsicher geltende Methoden ist bei der Implementierung so weit als möglich zu verzichten. Anschließend empfiehlt es sich, den entstandenen Quellcode von einer unabhängigen Instanz auf Schwachstellen hin untersuchen zu lassen, um diese daraufhin zu schließen. Dies kann beispielsweise mit der hier vorgestellten Methode geschehen. Abschließende Penetrationstest können weitere, bis dato unentdeckte, Schwächen aufdecken.

2.2 Sichtweise und Vorgehen eines Angreifers

Bei der manuellen Suche nach Schwachstellen ist es hilfreich, die Sichtweise und das mögliche Vorgehen eines Angreifers im Hinterkopf zu behalten.

Ziel eines Angreifers ist es unberechtigt Daten zu lesen, schreiben oder auszuführen. Daten schreiben beinhaltet dabei auch das Löschen. Daten ausführen ist mit der Ausführung von Programmcode gleichzusetzen. Er versucht also seine Rechte horizontal oder vertikal auszuweiten. Ein weiteres Ziel kann es sein, die Erreichbarkeit zu be- oder verhindern, indem er andere daran hindert, Daten zu lesen, schreiben oder auszuführen.

Um seine Ziele zu verfolgen, sind verschiedene Punkte interessant. Zum einen die möglichen Einstiegspunkte. Das sind Stellen, an denen er Daten in die Anwendung einbringen oder sonst wie auf sie einwirken kann. Zum anderen sind das die Angriffsziele, also Bereiche in denen eine Rechteausweitung erfolgen oder die Erreichbarkeit gestört werden kann, indem er das Programm zum Absturz bringt oder verfügbare Ressourcen, wie etwa Speicherplatz, erschöpft, sodass für andere Institutionen keine mehr zu Verfügung stehen.

Ein Angreifer kann über mehrere Wege versuchen Schwachstellen aufzudecken.
Zum einen kann er von den Einstiegspunkten ausgehend den Datenfluss verfolgen. Findet er Möglichkeiten im Zielbereich unverifizierte Eingaben schadhaft einzubringen, so hat er sein Ziel erreicht. Dabei können diese auch in Dateien, wie etwa in Konfigurationsdateien, zwischengespeichert und später an anderer Stelle, eventuell mit anderen Rechten, wieder aufgerufen werden. Sobald die Eingaben derart verifiziert werden, dass er damit keinen schadhaften Einfluss mehr ausüben kann, so bricht er die Untersuchung ab.
Zum anderen kann er versuchen, im Zielbereich Schwächen zu finden, um ausgehend von diesen den Datenfluss rückwärts zu folgen. Führt dieser zu Einstiegspunkten, und werden die Daten auf dem Weg nicht ausreichend überprüft, so hat er eine ausnutzbare Schwachstelle gefunden.

Neben dem Entdecken von Schwächen, stellt das Identifizieren der Angriffsziele und der Dateien, die für eine indirekte Kommunikation genutzt werden können, die besondere Herausforderung für den Angreifer dar.

2.3 First Principles Vulnerability Assessment

Die First Principles Vulnerability Assessment (FPVA) Methode ist ein Ansatz zur manuellen Beurteilung der Verwundbarkeit von Middleware in verteilten Systemen. First Principles bedeutet dabei, dass die Suche nach Schwachstellen vom Analyseprozess geleitet wird und nicht von bereits bekannten Schwachstellen. Bei der Durchführung der Methode werden daher vorab keine Annahmen über die Natur einer Bedrohung oder die Techniken, die angewendet werden um eine Schwachstelle auszunutzen, gemacht. Sie stellt ein so genanntes White Box Verfahren dar, setzt also den Zugang zu den Quelltexten voraus. Vorhandene Dokumentationen und ein Kontakt mit den Entwicklern kann hilfreich sein, ist aber nicht zwingend notwendig.

Die Methode arbeitet in fünf aufeinander folgenden Schritten. Eine manuelle Codeanalyse ist ein sehr aufwendiger Prozess und eine zeilenweise Analyse des gesamten Quellcodes ist aufgrund der Größe und Komplexität der meisten Anwendungen oft nicht praktisch durchführbar. Daher werden in den Schritten eins bis drei die dem System zugrunde liegende Architektur, das Interaktionsmodell, die verwendeten Ressourcen und die Behandlung von Rechten analysiert, um auf Basis der Ergebnisse Ziele und Zwischenziele mit hohem Schadpotential bzw. von großem Wert (hight value assets) zu identifizieren. Von diesen aus erfolgt im vierten Schritt die eingehende Untersuchung der einzelnen Komponenten, indem im Quellcode manuell nach möglichen Schwachstellen gesucht wird. Es wird also eine Abstraktionsebene eingeführt, mit deren Hilfe die Analyse geleitet wird. Die Schritte helfen ein besseres Verständnis für die Arbeitsweise der Anwendung zu erlangen, so dass es erleichtert wird, nicht nur Schwachstellen in den einzelnen Komponenten zu entdecken, sondern auch solche die durch die Zusammenarbeit der Komponenten entstehen. Diese Abhängigkeiten könnten bei einer Analyse rein auf Basis der einzelnen Komponenten eventuell unentdeckt bleiben. Schritt fünf ist die Veröffentlichung der Ergebnisse und eventuell gefundener Schwachstellen. Nachfolgend werden die fünf Schritte noch einmal detailliert beschrieben [3].

2.3.1 Architekturanalyse

Im ersten Schritt der Analyse mit der FPVA Methode werden mit Beschreibungen ergänzte Diagramme erstellt, die die Struktur des Systems sowie die Interaktionen der Komponenten untereinander, mit externen Anwendungen und mit Anwendern darstellen. Dazu werden zuerst die strukturellen Hauptkomponenten des Systems, wie Hosts, Module, Prozesse sowie Threats, ermittelt. Anschließend werden für die gefundenen Komponenten die Art der Kommunikation untereinander so wie mit externen Anwendungsprogrammen und Anwendern, also das Kommunikationsmodell, erruiert. Dies gibt einen ersten Überblick über die Arbeitsweise und die Implementierung der Software, was es im weiteren Verlauf erleichtert die Aufgaben von Funktionen zu verstehen, logische Fehler im Design zu entdecken und mögliche Einstiegspunkte für Angreifer zu identifizieren. Auch können dadurch im weiteren Verlauf einer Untersuchung einzelne Teilbereiche identifiziert werden, die besonders zu betrachten sind, weil mögliche Angreifer Einfluss auf sie haben könnten. Die gesuchten Informationen

können aus bestehenden Dokumentationen ermittelt werden. Diese sind aber möglicherweise unvollständig, veraltet oder schlichtweg falsch. Sie sollten daher nur als erste Anhaltspunkte genutzt werden. Eine weitere Quelle sind die Entwickler der Software, falls man Kontakt zu ihnen aufbauen kann. Diese haben allerdings keine unvoreingenommene Sicht auf das System und daher sollte als letzte Instanz immer der Quellcode der Software untersucht werden. Dies ist zwar die aufwendigste Methode an die gewünschten Informationen zu gelangen aber auch die zuverlässigste. Um die Komponenten zu identifizieren kann die Ordnerstruktur der Quellen bereits erste Anhaltspunkte bieten. Falls die Quelltexte in der Programmiersprache C verfasst sind, ist es hilfreich nach Funktionen wie exec(), system() oder popen() zu suchen, mit denen Prozesse gestartet werden können. Des weiteren kann nach dem fork() Befehl, welcher Kindprozesse erzeugt, und nach Funktionen zu erzeugen und kontrollieren von Threads gesucht werden. Im Bereich der Kommunikation sind beispielsweise Sockets, Pipes, Signale und Umgebungsvariablen gute Einstiegspunkte für die Untersuchung.

2.3.2 Identifizieren der Ressourcen

Schritt zwei der Methode ermittelt die Hauptressourcen auf die die Komponenten aus Schritt eins zugreifen sowie eventuelle Funktionen die die jeweiligen Ressourcen bereitstellen. Zu ihnen gehören z.B. Dateien, Datenbanken, Logs sowie Geräte. Da diese Ressourcen oft als Ziele für Angriffe dienen, wird für ausgewählte Ressourcen der Wert als mögliches Angriffsziel diskutiert. Dateien mit sensiblen Daten, wie Passwort Dateien, oder Datenbanken mit sensiblen Daten können dabei beispielsweise von höherem Wert für einen potentiellen Angreifer sein als Logdateien. Der Fokus der weiteren Untersuchung kann hierdurch bereits auf besonders schützenswerte Ressourcen gelenkt werden, wobei darauf zu achten ist, sich nicht zu sehr auf diese zu versteifen und dadurch andere Schwachstellen zu übersehen. Die Diagramme aus Schritt eins werden nun um die Informationen über die Ressourcen ergänzt bzw. es werden weitere Diagramme erstellt, die die Ressourcen beinhalten.

2.3.3 Vertrauen und Rechte

Im letzten Schritt vor der eigentlichen Auswertung, also im Schritt drei, werden die Themen Vertrauen und Rechte behandelt. Also zum Einen wie die jeweiligen Komponenten vor fremdem Zugriff und Veränderungen geschützt sind und zum Anderen mit welchen Rechten sie ausgestattet sind. Um zu entscheiden wie sich eine Komponente im Zusammenspiel mit einer Anderen verhält, wird in diesem Schritt zusätzlich die Weitergabe (Delegation) der Rechte untersucht. Mit der Betrachtung der Rechte kann der Schaden der bei einem erfolgreichen Angriff auf die Komponente entsteht abgeschätzt werden. Ein erfolgreicher Angriff auf eine Komponente die mit Root Rechten läuft kann erheblich mehr Schaden anrichten als auf eine die mit eingeschränkten Rechten ausgestattet ist. Mit Hilfe der gewonnenen Erkenntnisse kann also vorhergesagt werden wo nach Schwachstellen die eine vertikale Rechteausweitung, also das Erlangen von mehr Rechten, und wo nach Möglichkeiten der horizontalen Rechteausweitung, also das unbefugte Erlangen von Rechten anderer Benutzer, zu suchen ist. Die Ergebnisse finden wieder Einzug in die bereits vorhanden Diagramme. Es werden die jeweiligen Rechte vermerkt und die Diagramme um die Interaktionen, die im Zusammenhang mit der Delegation von Rechten stehen, erweitert.

2.3.4 Auswertung der Komponenten (manuelle Codeanalyse)

Nun folgt die eingehende Auswertung der Komponenten, die manuelle Code-Analyse. Dabei werden mit Hilfe der Ergebnisse aus den ersten drei Schritten zuerst einzelne Angriffsziele identifiziert und priorisiert und daraufhin der Quellcode, ausgehend von den Zielen mit den höchsten Wert für einen Angreifer, detailliert nach Schwachstellen untersucht. Wertvoll kann dabei für einen Angreifer zum Beispiel das Erlagen von sensiblen Informationen, unerlaubte Nutzungsmöglichkeit von Ressourcen oder das Schädigen des Rufs eines Unternehmens, indem etwa die Erreichbarkeit des Dienstes unterbunden wird, sein. Die Priorisierung hilft bei begrenzten Mitteln, Personal und Budget, den Aufwand bei der Suche nach Schwachstellen zu reduzieren, da erfolgreiche Angriffe auf Ziele ohne hohen Wert nur einen geringen Schaden verursachen und die Suche danach aufgeschoben werden kann. Die manuelle Codeanalyse kann durch Programme, die selbstständig nach Schwachstellen suchen, unterstützt werden. Diese haben aber auch Nachteile auf die in Kapitel 3 3.2 genauer eingegangen wird. Gefundenen Schwachstellen werden schließlich in „vulnerability reports“ beschrieben und diese können den Software Entwicklern zum beheben übermittelt werden.

2.3.5 Veröffentlichung der Ergebnisse

Vor der Veröffentlichung der Ergebnisse müssen einige Fragen beantwortet werden. Zum Beispiel in welchem Umfang und zu welchem Zeitpunkt dies geschehen soll. Entwickler brauchen Zeit um die Lücken zu schließen und die Modifikationen zu veröffentlichen. Können Anwender bis dahin im unklaren gelassen werden oder geben vorab veröffentlichte Ergebnisse möglichen Angreifern Hilfestellungen, bis ein möglicher Fix veröffentlicht wurde? Sind diese Punkte geklärt, können die Ergebnisse veröffentlicht werden.

2.4 Globus Toolkit 5 (GT5)

Das Globus Toolkit ist eine von der Globus Alliance, eine Gemeinschaft verschiedener Universitäten und Forschungszentren welche Technologien, Standards und Systeme im Grid Umfeld erforscht und entwickelt[8], implementierte und zu Verfügung gestellte Open Source Grid Middleware. In seiner aktuellen Version 5.02 bietet das Globus Toolkit Komponenten an, die den Aufbau von Grid Systemen und die Entwicklung von Grid Anwendungen unterstützen. Dabei beinhaltet es Software und Bibliotheken für die Bereiche Sicherheit, Datenverwaltung und -übertragung sowie für das Überwachen, Erkunden und Verwalten von Ressourcen. Grafik 2.1 zeigt die vom Globus Toolkit 5 bereitgestellten Komponenten [9]. Einzelne Komponenten nutzten dabei teilweise Dienste die von Anderen über APIs zu Verfügung gestellt werden.

Das Globus Toolkit muss nicht als Ganzes installiert werden, sondern es können einzelne Teile, sofern keine Abhängigkeit von anderen Komponenten besteht, unabhängig voneinander installiert und genutzt werden. Im Folgenden werden die einzelnen Komponenten vorgestellt. Die Beschreibungen lehnen sich dabei an die von der Globus Alliance veröffentlichten Dokumentationen zum Globus Toolkit 5 [10] an.

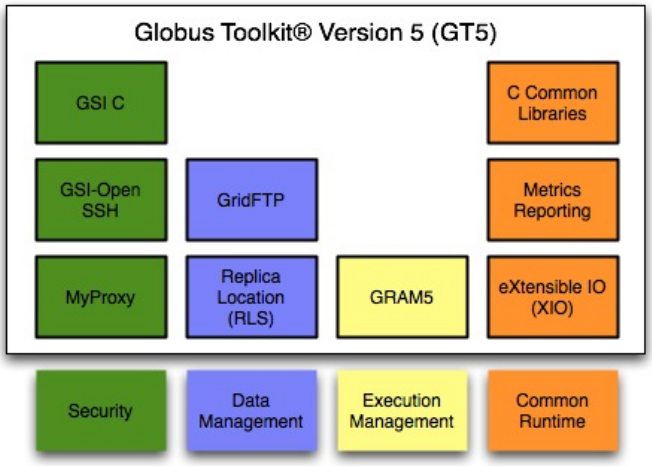

Abbildung 2.1: Komponenten des Globus Toolkit 5[9]

2.4.1 Common Runtime Components

C Common Libraries: Die Common Runtime Komponenten beinhalten die zum Aufbau einer Grid Umgebung benötigten C Standard Bibliotheken (*C Common Libraries*). Diese bieten eine Abstraktionsschicht für Datentypen, Systemaufrufe und Datenstrukturen, welche im Globus Toolkit genutzt werden.

Globus XIO: Ein weiterer Bestandteil ist eine in C geschriebene erweiterbare Input/Output Bibliothek (*Globus XIO*) die den dynamischen Ansprüchen einer Grid Anwendung gerecht wird. XIO stellt für verschiedene IO Protokoll-Treiber über das Globus XIO Framework eine einheitliche Programmierschnittstelle (API) zu Verfügung. Treiber können einzeln oder gestapelt (Driverstack) verwendet werden was die Wiederverwendbarkeit von Quellcode verbessert. Das Framework verwaltet dabei Anfragen die über die API gestellt werden und bildet sie auf einen Driverstack ab. Der unterste Treiber im Stapel ist dabei immer ein Transport-Treiber, der für die eigentliche Übertragung der Daten zuständig ist. Darüber können mehrere Treiber zum manipulieren der Daten, sogenannte Transform-Treiber, zum Einsatz kommen. Diese können zum Beispiel für die Komprimierung zuständig sein oder die Daten mit einer für die Sicherheit zuständigen Schicht umgeben wie der GSI-Treiber. XIO benutzt ein asynchrones Eventmodell, dabei werden Lese- und Schreibanfragen mit einem vom Anwender vorgegebenen Funktionszeiger versehen und bei Beendigung der Aktion wird die jeweilige Funktionen, die Callbackfunktion, aufgerufen. Zur Entwicklung von eigenen Treibern beinhaltet XIO eine eigene Entwicklungsschnittstelle. Das Globus Toolkit 5 hält, unter anderen, bereits TCP-, HTTP- und GSI-Treiber bereit. Da XIO die unterschiedlichsten Protokoll-Treiber unterstützt, hängt die Sicherheit dieser Komponente im Wesentlichen von den verwendeten Treibern ab.

2.4.2 Security Components

Zum Betrieb einer Grid-Infrastruktur wird eine sichere, d.h. authentifizierte und evtl. verschlüsselte, Kommunikation benötigt. Da auf Grund der verschiedenen Sicherheitsrichtlini-

en der einzelnen Virtuellen Organisationen keine zentral verwaltete Sicherheitsinfrastruktur zum Einsatz kommen kann, nutzt die Grid Sicherheits Infrastruktur (GSI) als Basis asymmetrische Verschlüsselung auf Grundlage von Public Key Infrastrukturen (PKI) [11] zur Authentifizierung von Kommunikationspartnern. Mit deren Hilfe wird auch „Single sign-on“ und ein Delegationsmechanismus, der beispielsweise benötigt wird wenn eine Anfrage mehrere Ressourcen braucht oder externe Daten beschaffen muss, realisiert. Bei einer PKI werden digital signierte Zertifikate (hier X.509 Zertifikate), mit Unterstützung von Transport Layer Security (TLS), verwendet um die Identität der jeweiligen Kommunikationspartner durch eine wechselseitige Authentifizierung (Mutual Authentication) zu belegen. Zum delegieren von Rechten werden Proxy-Zertifikate [12] eingesetzt. GSI schützt die Kommunikation zwar durch digitale Signaturen vor Manipulationen (Integrität), ein Verschlüsselungsmechanismus ist aber auf Grund der US Export Gesetzte nicht standardmäßig enthalten. Mit Hilfe der verwendeten PKI kann aber leicht ein sicherer Schlüsselaustausch erfolgen um Vertraulichkeit zu realisieren.

GSI C: Zur Realisierung der oben beschriebenen Funktionalitäten stellt das Grid Security Infrastuctur in C (GSI C) Framework mehrere APIs und Hilfsprogramme (Tools) bereit. Eine API für die Authentifizierung, eine erweiterte Instantiierung der Generic Security Services-API (GSSAPI)[13, 14, 15] basierend auf Transport Layer Security mit einer erweiterten Pfadvalidation um mit Proxyzertifikaten umgehen zu können, sowie zwei für die Autorisierung. Die GSSAPI wird von mehreren APIs, die sich um die Details der Authentifizierung kümmern, unterstützt. Für die Autorisierung enthält GSI C die generische Autorisierungs API und die Gridmap API. Erstere stellt ein Framework zu Verfügung, das Anfragen an Drittanbietersoftware erlaubt und basiert auf Subjekt-Aktion-Objekt Tupeln die bestimmen wer eine bestimmte Aktion auf einer bestimmten Ressource ausführen darf. Die Zweite, die Gridmap API, erlaubt auch das standartmäßige Verhalten durch einfügen von Anfragen zu verändern. Sie hat aber einen voreingestellten Mechanismus bei dem auf Basis von Gridmapfiles die Subjekte der Zertifikate an lokale Benutzer, z.B. lokale UNIX User, gebunden werden. Einzelne Ressourcen sowie die Aktionen die auf ihnen ausgeführt werden sollen, werden bei dieser API nicht betrachtet.

MyProxy: Der MyProxy Credential Management Service, ist ein Programm zum Verwalten von X.509 PKI Sicherheits-Zertifikaten und privater Schlüssel, das hauptsächlich zum Verwalten von kurzlebigen Proxyzertifikaten genutzt wird. Die Software stellt für diesen Zweck ein online Archiv für Zertifikate zu Verfügung. Proxyzertifikate können im Archiv gespeichert werden und im Delegationsfall muss nur die MyProxy Passphrase an den Service weitergeben werden. Auch eine online Zertifizierungsstelle (CA) stellt MyProxy bereit. Damit können Zertifikate angefordert und verlängert, sowie Zertifikatsperrlisten abgefragt und Authentifizierungsprozesse durchgeführt werden.[16]

SimpleCA: *SimpleCA* [17]ist eine Software die die OpenSSL CA[18] Funktionalität zum erstellen einer Zertifizierungsstelle erweitert, um Grid spezifische Anforderungen zu erfüllen. Die Software ist sehr einfach. Sie sollte nur zu Testzwecken eingesetzt werden oder falls keine andere Zertifizierungsstelle zu Verfügung steht.

GSI-OpenSSH Das Glotbus Toolkit 5 liefert auch eine erweiterte Version von OpenSSH[19], *GSI-OpenSSH*. GSI-OpenSSH unterstützt, zusätzlich zu den von OpenSSH angebotenen Mechanismen, die Authentifizierung und die Delegation von Rechten mittels X.509 Proxyzertifikaten.[20]

2.4.3 Data Management Components

Im Bereich der Datenverwaltung biete das Globus Toolkit 5 mit GridFTP und dem Replica Location Service (RLS) Komponenten, die sich um das Speichern, den Transfer sowie die Verwaltung von verteilten Daten und eventueller Kopien kümmern.

GridFTP: *GridFTP* [21, 22] ist eine Erweiterung des File Transfer Protokolls (FTP)[23] und ermöglicht den sicheren, zuverlässigen, schnellen sowie effizienten Transfer von Daten. In Verbindung mit Grid Security Infrastructure unterstützt es X.509 Proxy Zertifikate zur Authentifikation. Auch kann mit GridFTP ein Datentransfer zwischen verschiedenen Servern mittels des Clients ferngesteuert werden. Zum schnellen Datentransfer ermöglicht GritFTP das simultane Übertragen über verschiedene Verbindungen. Dadurch kann eine Datei gleichzeitig von verschiedenen Quellen oder parallel über mehrere Verbindungen von der gleichen Quelle geladen werden. Zur Kontrolle von GridFTP dienen mehrere Kommandozeilenprogramme, wie globus-url-copy, und für die Entwicklung stehen die Globus FTP Client Library und die Globus FTP Library APIs zu Verfügung. Der Steuerkanal von GridFTP ist standardmäßig verschlüsselt wohingegen der Datenkanal per Grundeinstellung nur authentifiziert wird und eine Integritätsprüfung der Daten sowie eine Verschlüsselung explizit aktiviert werden müssen.

Replica Locaton Service: Der *Replica Location Service (RLS)* ist ein einfaches Register, welches logische Dateinamen auf physische Speicherorte abbildet. Mit seiner Hilfe können verteilte Datensätze und Kopien geortet werden. Anwender übermitteln beim Erzeugen und Verschieben von Dateien oder Duplikaten die entsprechenden Informationen über den Speicherort an den Replica Location Service, der die Informationen in einer Datenbank speichert. Die Einträge können von autorisierten Nutzern, mit Hilfe der RLS Client C API, abgefragt werden. Der Replica Location Service selbst besteht aus zwei Teilen, dem Location Replica Catalog (LRC) Server und dem Replica Location Index (RLI) Server. Ein oder mehrere LRC Server speichern dabei die eigentliche Abbildung der Dateinamen auf die Speicherorte und informiert den RLI Server über Einträge. Anfragen werden an den RLI Server gestellt, der in der Antwort auf die entsprechenden LRC Server verweist. Der Replica Location Service überprüft Kopien nicht auf ihre Gültigkeit, also ob eine Kopie korrekt bzw. konsistent ist. Zur Administrierung und für diverse Operationen auf die im RLS gespeicherten Informationen existieren diverse Kommandozeilenprogramme.

2.4.4 Execution (Job) Management Components

Grid Resource Allocation and Management 5: Für die Vermittlung von Jobs an Ressourcen sowie für die Überwachung, Manipulation und den Abbruch von Jobs steht die Grid Resource Allocation and Management 5 (GRAM5) Service Komponente zu Verfügung. GRAM nützt eine Austauschsprache um Jobanfragen zu formulieren und Ressourcen zu beschreiben, die Globus Resource Specification Language (RSL). Die Kommunikation zwischen Clients

und dem Service erfolgt über das Gram Protocol, welches auf einer Untermenge des HTTP 1.1 Protokolls basiert. GRAM5 authentifiziert die Kommunikationspartner wechselseitig und überprüft ob diese berechtigt sind die angefragten Ressourcen zu nutzen. Dies geschieht unter Zuhilfenahme der Grid Security Infrastruktur im globus-gatekeeper, ein Bestandteil des GRAM Service. Nach erfolgreicher Autorisierung startet der Gatekeeper den eigentlichen Job Manager Service, den globus-job-manager, und reicht die Anfrage zur weiteren Bearbeitung an diesen weiter. Die darauf folgende Kommunikationen mit den Clients, zum Beispiel bei Statusanfragen, Benachrichtigungen bei Statusänderungen und Jobabbrüche, erfolgen direkt mit dem Job Manager über den vom Gatekeeper aufgebauten Security Context. Der Job Manager kümmert sich, mit Hilfe der Daten Management Komponenten, um den Transfer von Dateien und leitet die Jobanfrage über das Job Manager Skript und den Job Manager LRM Adapter an den zuständigen Local Resource Manager (LRM) weiter. Der Job Manager LRM Adapter wandelt die Anfrage dabei in ein Local Resource Managern kompatibles Format um. Die Local Resource Manager verwalten die Ressourcen und übernehmen das eigentliche Scheduling der Jobs. Sie sind kein Bestandteil von GRAM5. GRAM5 ist also kein Resource Scheduler sondern eine Art Front-End, das die Kommunikation mit verschiedenen Local Resource Managern ermöglicht sowie Verwaltungsaufgaben übernimmt. Informationen über den aktuellen Status eines Jobs kann der Job Manager mit Hilfe des Scheduler Event Gernerators erlangen oder regelmäßig von den Local Resource Managern erfragen. Beim Einsatz des Scheduler Event Generators übersetzen Scheduler Event Generator LRM Module dabei die Informationen, die von den unterschiedlichen Local Resource Managern in Logfiles gespeichert werden, in ein für GRAM5 verständliches Format und informieren den zuständigen Job Manager. Dieser leitet die Informationen an Clients die spezielle Statusupdates abonniert haben oder als Reaktion auf explizite Anfragen weiter und kann diese zusätzlich für Abrechnungszwecke in einer Audit Datei speichern. Auf der Client Seite existieren die Gram Protokoll-, RSL- und Client-API sowie etliche Kommandozeilenprogramme, für Job- und Statusanfragen, Jobabbruch etc. Unter Zuhilfenahme von Local Resource Managern wie Condor[24], die auch als Clients fungieren können, ist es möglich eine mehrschichtige Architektur aufzubauen und so einen Meta-Scheduler bereitzustellen.

Unterschiede zum Globus Toolkit 4

Das Globus Toolkit 5 weist im Vergleich zum Globus Toolkit 4 einige eklatante Änderungen auf. Während die meisten Komponenten wie GridFTP, RLS und MyProxy nur geringfügige Veränderungen in Form von Fehlerbehebungen und neuen Funktionen erfahren haben, fallen andere ganz weg und werden durch Neue ersetzt. Dazu gehören der GT4 Java Core, das Reliable Transfer Service (RFT) und das Web Services Grid Resource Allocation and Management (WS-GRAM4). Am auffälligsten dabei ist der Wegfall von WS-GRAM4, das durch GRAM5 abgelöst wurde. GRAM4 hat die Anforderungen die eine Gridumgebung stellt durch das Bereitstellen einer zum Web Service Resource Framework (WSRF) [25] konsistenten Web Service [26] Schnittstelle erfüllt. Dies führte oft zu Problemen in der Ausfallsicherheit. Daher wurde GRAM2 (pre WS-GRAM), das auf statusbehafteten Ressourcen beruht aber Probleme mit der Skalierbarkeit hat, weiterentwickelt. Das Ergebnis ist GRAM5, das nach Aussagen der Entwickler ausfallsicher und skalierbar arbeitet. Die Funktionalität des Reliable Transfer Service wurden in die GRAM5 Komponente integriert [27].

2.5 Zusammenfassung

Der Begriff Verwundbarkeit bezeichnet die Existenz von Schwachstellen in einer Software welche durch Schadprogramme ausgenutzt werden können.

Bei der Suche nach Schwachstellen sollten die Ziele und das mögliche Vorgehen eines Angreifers berücksichtigt werden.

Mit Hilfe der First Principles Vulnerabiltiy Assessment Methode kann in fünf klar definierten Schritten nach Schwachstellen in verteilten Anwendungen gesucht und so die Verwundbarkeit beurteilt werden. Erst erfolgen die grundlegenden Analysen der Architektur, der zu den Komponenten gehörenden Ressourcen und der Rechte mit denen die Komponenten ausgestattet sind, bevor die eigentliche manuelle Codeanalyse, ausgehend von den durch die ersten drei Schritte bestimmten Zielen, durchgeführt wird. Die Methode verfolgt also den in der Informatik oft genutzten Ansatz zu abstrahieren um die Komplexität zu verdecken und den Fokus gezielt richten zu können, um anschließend ausgewählte Bereiche gezielt zu betrachten. Abschließend werden die Ergebnisse veröffentlicht.

Das Globus Toolkit 5 stellt Komponenten in den Bereichen Common Runtime, Sicherheit, Datenmanagement und Ausführungsmanagement zur Verfügung mit deren Hilfe eine komplexe Grid Infrastruktur instantiiert werden kann. Der größte Unterschied zum Globus Toolkit 4 liegt in einem neuen Ressource und Allocation Management, GRAM5, das nicht mehr auf Web Services sondern auf statusbehafteten Ressourcen beruht.

3 Verwandte Arbeiten

In diesem Kapitel werden exemplarisch einige Arbeiten vorgestellt die sich mit der Suche nach Schwachstellen im Allgemeinen und mit der First Principle Vulnerability Assessment Methode im Speziellen befassen. Viele Projekte, wie das Open Web Application Security Project (OWSAP) [28] oder die National Vulnerability Database [29], stellen Bibliotheken mit bekannten Schwachstellen und Angriffen sowie Sammlungen von Best Practice Prozessen bereit. Sie sollen helfen, sichere Software zu entwickeln bzw. die Sicherheit bestehender Anwendungen zu erhöhen. Die Beispiele in diesem Kapitel zeigen, wie wichtig mittlerweile das Thema Schwachstellenanalyse in Wirtschaft und Forschung geworden ist.

3.1 Threat Modeling

Threat Modeling ist eine von Microsoft propagierte und in ihrem Secure Development Lifecycle angewandte Methode, um mögliche Bedrohungen (threats) eines Systems zu identifizieren, klassifizieren und dokumentieren [30, 31]. Sie wird auch von dem Open Web Application Security Project, in seiner Best Practice Sammlung zur sicheren Entwicklung von Software, dem Comprehensive Lightweight Application Security Process (CLASP), als Methode zur Sicherheitsanalyse vorgeschlagen. Threat Modeling ist ein iterativer Prozess, welcher während der Design- und Implementierungsphase des Systems angewandt wird, bis eine gewünschte Detailstufe erreicht ist. Um Risiken im Code abzuschwächen und Entscheidungshilfen bei der weiteren Entwicklung zu geben, werden die Ergebnisse nach jedem Iterationsschritt an die Entwicklungsteams weitergegeben. Unabhängig davon kann die Methode auch genutzt werden, um fertige Programme zu untersuchen.

Die Schritte der Threat Modeling Methode im Einzelnen:

1. Identifizieren der wertvollen Ressourcen (high value assets), also der Ziele die besonders schützenswert sind.

2. Erstellen eines einfachen Architekturdiagramms, mit Fokus auf die identifizierten Ressourcen, um einen Überblick über das System zu erhalten und zu sehen, wie die wertvollen Ressourcen genutzt werden.

3. Detailliertere Betrachtung der Architektur, wobei besonderes Augenmerk auf Vertrauensgrenzen, Einstiegspunkte, Datenflüsse und die Rechte, unter denen Komponenten laufen, gelegt wird.

4. Identifizieren von Bedrohungen die eine Gefahr für die in Schritt eins identifizierten Ressourcen darstellen. Dies geschieht entweder durch Anwendung der STRIDE Methode 3.1 oder über kategorisierte Listen von bekannten Bedrohungen, wie sie beispielsweise von OWSAP bereitgestellt werden. Anschließend werden die einzelnen Bedrohungen

mittels DREAD 3.1 bewertet, um zu entscheiden wie viel Aufmerksamkeit den jeweiligen Bedrohungen im Weiteren zugestanden wird.

5. Suche nach Sicherheitslücken. Dabei werden den identifizierten Bedrohungen bekannte Sicherheitslücken zugeordnet, nach denen in den Quelltexten gesucht wird.

STIDE ist eine von Microsoft entwickelte Methode um Bedrohungen zu identifizieren [32]. Als Akronym steht es für:

- Spoofing identity: Vortäuschen einer falschen Identität.
- Tampering with data: Unautorisiertes Verändern von Daten.
- Repudiation: Verschleiern von durchgeführten Angriffen.
- Information disclosure: Unerlaubtes Erlangen von vertraulichen Daten.
- Denial of Service: Verfügbarkeit eines Dienstes verhindern.
- Elevation of privilege: Unerlaubtes Erlangen von Rechten.

Um die einzelnen Bedrohung zu Bewerten wird das auch von Microsoft entwickelte DREAD Verfahren angewandt [31]. Bedrohungen werden dabei in fünf Kategorien eingeteilt, um die Bedrohungen zu priorisieren. DREAD ist wiederum ein Akronym und steht für:

- Damage potential: Wie hoch ist der Schaden der entstehen kann?
- Reproducibility: Wie leicht kann der Vorgang nachgestellt werden?
- Exploitability: Wie leicht kann ein Angriff durchgeführt werden?
- Affected Users: Welche und wie viele Nutzer sind betroffen?
- Discoverability: Wie leicht kann die Bedrohung gefunden werden?

Zur Unterstützung der Threat Modelling Methode bietet Microsoft das SDL Threat Modeling Tool an [33].

Im Unterschied zur First Principles Vulnerabiliy Assessment Methode wird der Analyseprozess nicht durch die Methode bestimmt, sondern durch die zu Anfang identifizierten Ressourcen. Unter Umständen entziehen sich dadurch mögliche Zwischenziele der Untersuchung, da Zusammenhänge übersehen werden können. In der First Principles Vulnerabiliy Assessment Methode wird hingegen die ganze Anwendung betrachtet und erst in Schritt 3 werden die Ressourcen mit hohem Wert als mögliche Ziele identifiziert. Ein weiterer Nachteil der Threat Modelling Methode ist, dass sie ausschließlich vorab definierte und bereits bekannte Bedrohungen und Schwachstellen betrachtet. Neue Schwachstellen können durch die Methode, im Gegensatz zur First Principles Vulnerability Assessment Methode, nicht entdeckt werden.

3.2 Programme zur automatischen Suche nach Schwachstellen (Vulnerability Scanner)

Es existieren etliche kommerzielle und open-source Programme mit denen nach Schwachstellen in vorhandenem Quellcode oder in lauffähigen Programmen gesucht werden kann.

Erste führen eine statische Quellcodeanalyse durch, gehören also zu den White-Box Verfahren. Je nach Funktionsumfang melden sie mögliche Gefahrenstellen, wie die Benutzung von unsicheren Funktionen, die Verwendung nicht geprüfter Eingabedaten oder mögliche Buffer Overflow Schwachstellen. Ein Vorreiter in diesem Bereich war lint[34], sein Nachfolger ist Splint [35]. Weitere Beispiel sind das Rough Auditing Tool for Security (RATS) [36] und flawfinder [37]. Die hier aufgeführten Programme sind aber nicht in der Lage komplexe Abhängigkeiten von zusammenarbeitenden Komponenten, und damit Schwachstellen die aus dem Design resultieren, zu erkennen.

Zweite sind Vertreter der so genannten Black-Box Verfahren und untersuchen lauffähige Systeme. Sie führen eine dynamische Analyse ohne Überprüfung der Quelltexte durch. Bekanntester Vertreter dieser Black-Box Verfahren dürfte wohl der Nessus Vulnerability Scanner sein [38]. Er basiert auf einer Client/Server Struktur und kann über Netzwerkverbindungen Schwachstellen auf dem zu scannenden System entdecken. Der Funktionsumfang kann durch die Installation von Modulen bestimmt und erweitert werden.

Alle hier aufgeführten Programme arbeiten auf Basis bekannter Schwachstellen bzw. Angriffsmuster und können daher keine neuen Schwachstellen entdecken. Ausnahmen davon bilden Programme die auf dem Fuzz Testing Verfahren beruhen [4]. Diese erzeugen große Mengen an Zufallsdaten, die sie der zu untersuchenden Anwendung übergeben und versuchen so Schwächen aufzudecken. Eine exakte Lokalisierung der Schwachtellen in den Quelltexten ist aber kaum möglich.

Mittlerweile gibt es eine Unzahl von Programmen zur automatischen Suche nach Schwachstellen [39]. Allen zu Eigen ist, dass sie oft schwerwiegende Fehler nicht erkennen oder viele False-Positives liefern, also Fehler melden obwohl diese keine sind. Sie eignen sich daher nur bedingt zur eigenständigen Untersuchung oder zur Unterstützung von manuellen Analysen. [40].

3.3 Bereits mit der First Principles Vulnerability Assessment Methode untersuchte Systeme

Die Universitäten von Wisconsin und Barcelona haben mit der von ihnen entwickelten Methode bereits einige System im Gridumfeld untersucht. Dazu gehören zum Beispiel MyProxy und Condor.

MyProxy ist ein Managementsystem für Zertifikate und wird auch mit dem Globus Toolkit angeboten (vergleiche Abschnitt 2.4.2). Gegenstand der Analyse war die Version 4.2. Bei seiner Untersuchung wurde der MyProxy Server als kritisches Ziel identifiziert, da er die zu verwaltenden Zertifikate und die zugehörigen Schlüssel beherbergt. My Proxy zeigte bei

der Untersuchung fünf Schwachstellen, die auf kleinere Fehler im Design beruhen. Sie lassen sich nicht zum unrechtmäßigen Erlangen von Rootrechten ausnutzen. Drei der gefundenen Schwachstellen ermöglichen es durch Denial of Service Attacken die Verfügbarkeit des Servers zu stören. Sie können aber nicht das komplette System lahm legen und sind daher nicht von ernster Natur. Eine Schwachstelle rührt von der Tatsache her, dass keine Timeouts für eine Verbindung angegeben und die Größe der empfangenen Anfragen nicht begrenzt wurde. Dadurch kann eine große Zahl von Kindprozessen erzeugt werden. Ausserdem ist es möglich, verschiedene Obergrenzen für die vom Betriebssystem bereitgestellte Ressourcen zu erreichen. Eine Weitere entsteht dadurch, dass zum speichern von Zertifikaten ein einziges Verzeichnis verwendet wird. Legt ein Angreifer, der einen authentifizierten Zugriff zum System hat und berechtigt ist Zertifikate zu speichern, eine große Zahl an Zertifikaten in dem Verzeichnis ab, so kann sich die Antwortzeit des Systems erheblich verlängern. Die Analysten kommen zu dem Schluss, dass MyProxy sicher zu sein scheint und keine ersichtlichen, kritischen Schwachstellen enthält.

Condor, das auch im Globus Toolkit als Local Resource Manager zum Einsatz kommt, ist ein Workload Management System für rechenintensive Jobs. Mit seiner Hilfe kann auch eine mehrschichtige Architektur aufgebaut werden, um Jobs auf verschiedene Systeme zu verteilen. Dadurch kann ein Metascheduling vorgenommen werden. Bei der Untersuchung von Condor in der Version 6.7 zeigten sich mehrere kritische Design- und Implementierungsschwachstellen die es ermöglichen das komplette System zu kompromittieren. Insgesamt wurden fünfzehn Schwachstellen gemeldet, wovon dreizehn es erlauben Administrationsrechte zu erlangen und eine den Zugriff auf Root Accounts ermöglicht. Gefunden wurden unter anderem Directory Traversal, Command Injection, Log Injection und Buffer Overflow Schwachstellen.

Zwar dokumentieren die im Rahmen der Vorstellung der First Priciples Vulnerability Assessment Methode veröffentlichten Analysen [3] das grundsätzliche Vorgehen und die Ergebnisse, einen Rückschluss auf den Aufwand der betrieben wurde lassen sie aber kaum zu. Auch ist nicht ersichtlich, wie stark sich die Untersuchungen an die FPVA Methodik halten. Soll die Methode aber, wie von den Entwicklern gewünscht, Einzug in den Software Development Lifecycle halten, so kann eine unabhängige Bewertung die Akzeptanz erhöhen.

3.4 Zusammenfassung

Mit der Threat Modeling Methode existiert bereits ein Ansatz zur manuellen Codeanalyse, die sich zum Untersuchen verteilter Programme eignet. Sie basiert aber auf bereits bekannten Schwächen und kann neue nicht entdecken. Diese Lücke versucht die First Priciples Vulnerability Assessment Methode zu schließen. Obwohl etliche Programme zur automatischen Suche nach Schwachstellen existieren, entdecken diese schwerwiegende Fehler oft nicht und erzeugen viele False-Positives. Die veröffentlichten Untersuchungsergebnisse, der mit Hilfe der First Priciples Vulnerability Assessment Methode untersuchten Programme, stammen alle von den Entwicklern der Methode selbst und lassen nicht auf den Aufwand schließen, der zum Erlangen der Ergebnisse betrieben wurde. Deshalb eignen sie sich nicht, die Methode selbst zu bewerten und es ist sinnvoll eine unabhängige Untersuchung mit der Methode durchzuführen, um diese mit Hilfe der gewonnenen Erkenntnisse unabhängig zu bewerten.

4 First Principles Vulnerability Assessment angewandt auf GRAM5

In diesem Kapitel wird die First Principles Vulnerability Assessment Methode auf eine Komponente des Globus Toolkit 5 angewandt.

Gegenstand der Beurteilung ist der Globus Resource and Allocation Management Service in der Version 5.02 (GRAM5). GRAM5 bietet sich für eine Untersuchung an, da es sowohl mit den Client APIs, als auch mit den Local Ressource Managern kommuniziert. Des weiteren nutzt es zusätzliche APIs zum Authentifizieren der Kommunikationspartner, zum Autorisieren der Clients und kümmert sich um den Transfer von Daten. Die Untersuchung von GRAM5 stellt also eine Möglichkeit dar, den Umgang der Methode mit einer Vielzahl von genutzten APIs und Kommunikationspartnern zu untersuchen. Ein weiterer Grund für die Wahl dieser Komponente ist die große Veränderung, die sie im Vergleich zu ihrem Vorgänger (GRAM4) erfahren hat. Sie wendet sich vom Web Service Ansatz ab und erfüllt die Gridanforderungen, wie auch schon GRAM2, mittels statusbehafteter Ressourcen. Die daraus resultierenden grundlegenden Veränderung im Quellcode bieten, gerade in diesem frühen Versionsstand, viel Raum für Schwachstellen. Eine Beurteilung der Verwundbarkeit von GRAM in Version 5.02, auch wenn diese nur in Grundzügen verläuft, kann daher viel über den Umgang der Entwickler mit dem Thema Sicherheit aussagen und damit eine grundlegende Tendenz aufzeigen.

Im folgenden werden dazu die Schritte eins bis vier der First Principles Vulnerability Methode durchlaufen. Die Architekturanalyse, die Bestimmung der entscheidenden Ressourcen auf die die Komponenten zugreifen, die Analyse der Behandlung von Rechten und die Auswertung der Komponenten. Schritt fünf, die Veröffentlichung der Ergebnisse, erfolgt mit der Veröffentlichung dieser Arbeit. Abschließend werden die Ergebnisse zusammengefasst um GRAM5 bezüglich seiner Verwundbarkeit zu bewerten. Gemeinsam mit den bei der Durchführung gemachten Erfahrungen dienen sie auch dazu, die First Principles Vulnerability Assessment Methode selbst zu beurteilen.

4.1 Architekturanalyse

In der Architekturanalyse werden die Hauptkomponenten des Systems bestimmt. Zusätzlich erfolgt eine Analyse der Interaktionen zwischen ihnen, mit den Clients sowie mit externen Anwendungen. Die Informationen stammen zum Teil aus den Dokumentationen vom GRAM5, die auf der Projektseite des Globus Toolkits veröffentlicht sind [10] sowie aus einer ersten Sichtung der Quelltexte.

4.1.1 Komponenten

Die meisten Komponenten sind in der Sprache C geschrieben, nur das Globus Job Manager Skript und die Job Manager LRM Adapter Module sind in Perl implementiert.

Abbildung 4.1 zeigt eine Übersicht über die einzelnen Komponenten und gibt einen Überblick über die Interaktionen.

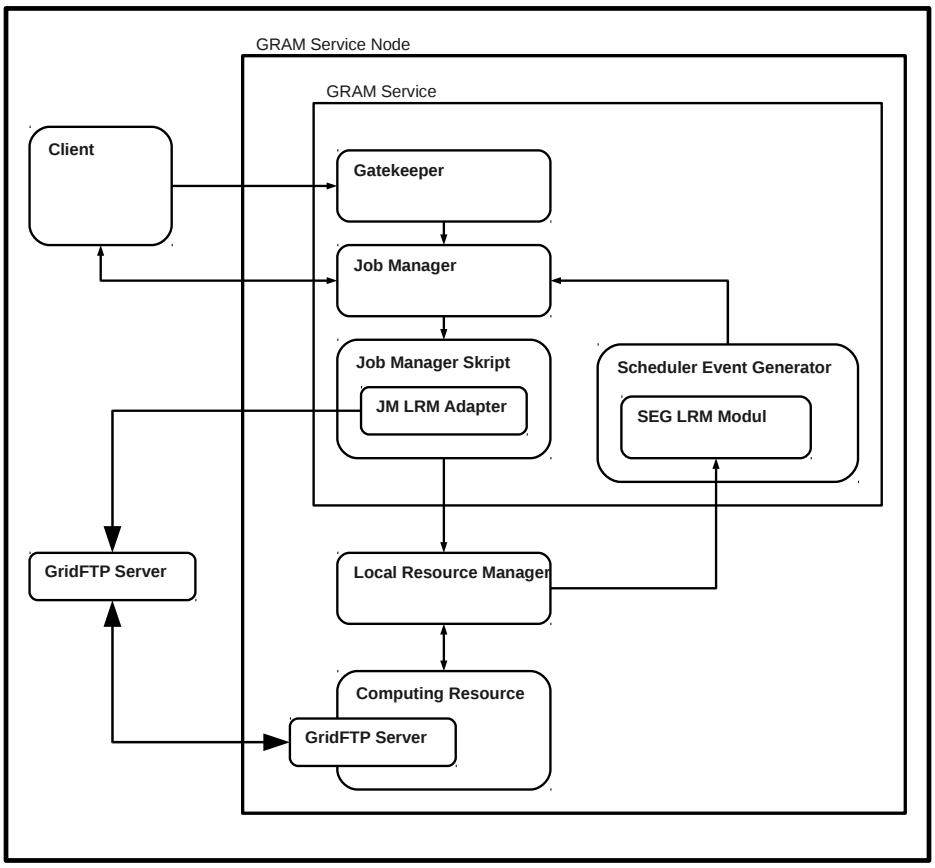

Abbildung 4.1: Übersicht über die Interaktionspfade von GRAM5

Client: Auf der Client Seite stellt GRAM5 mit der GRAM Client API Funktionen für die Jobübermittlung, zum Übertragen von Signalen und zum Anfragen von Statusupdates zu Verfügung. Diverse Kommandozeilenprogramme stehen für die Übermittlung von Jobanfragen, Abbrüchen, Statusanfragen und zum Absetzen anderer Anfragen für die Verwaltung von Jobs bereit. Beispiele hierfür sind globusrun, ein Programm zum übermitteln und verwalten von Jobs über GRAM sowie globus-job-status, mit dem der aktuelle Status eines Jobs erfragt werden kann. Des weiteren existieren die GRAM Protocol API und die Resource Spezifikation API mit deren Hilfe Nachrichten an die Server verfasst und ankommende Nachrichten entpackt werden können.

Gatekeeper: Der globus-gatekeeper Service ist ein Superserver wie inetd und stellt auf der Server Seite die zentrale Zugriffskomponente für initiale Jobanfragen dar. Wie in Abbildung 4.2 ersichtlich wird, kann er über einen Superserver wie inetd oder xinetd, der auf einem spezifizierten Port stellvertretend Verbindungen entgegen nimmt, bei jeder Anfrage neu gestartet werden. Auch kann er als Dämonprozess laufen, der zur Boot-Zeit initialisiert wird und bei jeder Jobanfrage, die an einen von ihm erstellten Socket gestellt wird, mittels fork() einen Kindprozess zur weiteren Bearbeitung der Anfrage erzeugt. Abbildung 4.3 stellt dies graphisch dar. Nach Eingang einer Anfrage führt der Gatekeeper die wechselseitige Authentifizierung der Kommunikationspartner durch und überprüft ob der Client zur Nutzung der angefragten Ressource autorisiert ist. Sind diese Schritte erfolgreich abgelaufen, so startet er einen Job Manager Prozess, übergibt ihm die Anfrage und terminiert dann, falls nicht explizit angegeben ist, dass er auf die Beendigung des Job Managers warten soll. Der Job Manager ist daraufhin für die weitere Bearbeitung der Aufträge zuständig.

Job Manager: Der Job Manager ist für die weitere Bearbeitung der Anfragen zuständig. Pro User/Local Resource Manager Kombination kann eine langlebige Instanz von ihm existiert bzw. vom Gatekeeper gestartet werden. Sie besteht so lange, wie für die jeweilige Kombination noch Jobs in Bearbeitung sind. Des weiteren gibt es pro Job eine kurzlebige Instanz. Die Job Manager Prozesse werden vom Gatekeeper erzeugt. Existiert noch keine langlebige Version für die entsprechende Kombination, so übernimmt der gestartete Prozess diese Aufgabe und forkt einen Kindprozess, der als kurzlebiger Job Manager fungiert und die weitere Bearbeitung übernimmt. Besteht bereits eine langlebige Instanz so fungiert der vom Gatekeeper erzeugte Prozess als kurzlebiger Job Manager und es wird kein Kindprozess erzeugt. Abbildung 4.4 zeigt die Schritte, die durchlaufen werden wenn bereits ein langlebiger Job Managern existiert. Abbildung 4.2 zeigt die Erzeugung eines Kindprozesses. Ein Job durchläuft während seiner Laufzeit verschiedene Zustände, wobei immer wieder eine Zustandsmaschine gestartet wird. Diese reagiert auf die einzelnen Status der Jobs, in dem sie beispielsweise das zwei Phasen Commit abwickelt, registrierte Clients über Statusänderungen informiert oder die Anfrage an das Job Manager Skript zur Weiterleitung an den LRM übergibt.

Globus Job Manager Skript: Das globus-job-manager-script.pl ist ein Perl Skript und arbeitet mit den Job Manager LRM Adaptern zusammen um Jobaufträge an die Locale Resource Manger zu übermitteln (submit), den Status von Jobs abzufragen (pull) oder diese abzubrechen (cancel). Es können zur Laufzeit bis zu fünf Instanzen pro User/LRM/Servicekonfiguration existieren. Dies wird bei der Initialisierung des Job Managers fest vorgegeben. Zu sehen in der Datei gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source/globus_gram_job_manager.c:

```
00321     /* Default number of scripts which can be run simultaneously */
00322     manager->script_slots_available = 5;
```

Um mit den Local Resource Managern zu kommunizieren nutzt das Skript das für den jeweiligen Manager zuständige Job Manager LRM Adapter Modul. Als Perl Modul wird es in das Job Manager Skript geladen und stellt die eigentliche Schnittstelle zwischen Job Manager und Local Resource Manager bzw. den GridFTP Servern dar, indem es die Anfragen

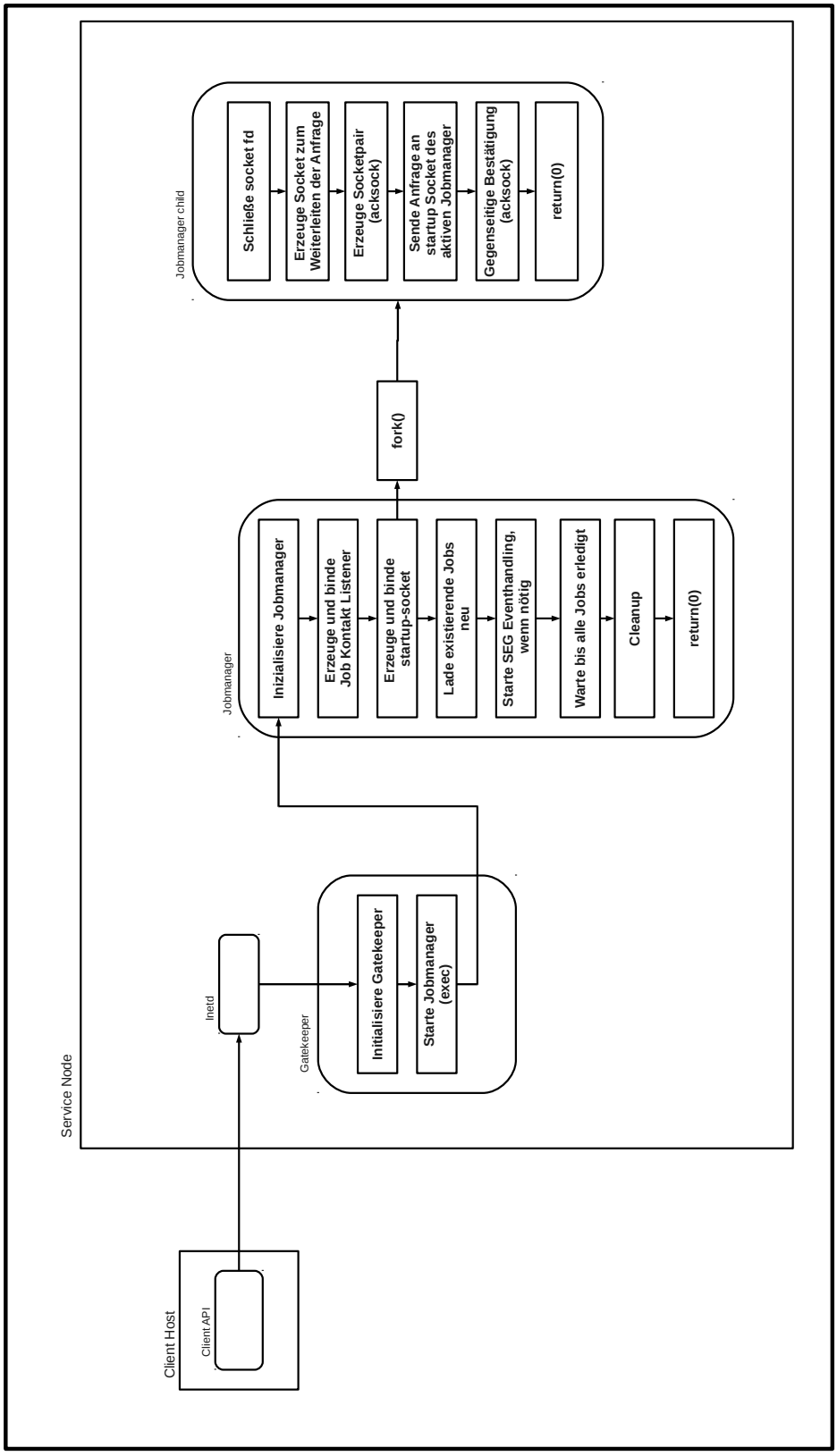

Abbildung 4.2: Interaktionen (Ein für eine bestimmte User/LRM Kombination zuständiger Job Manager existiert noch nicht. (Gatekeeper wird über inetd gestartet)

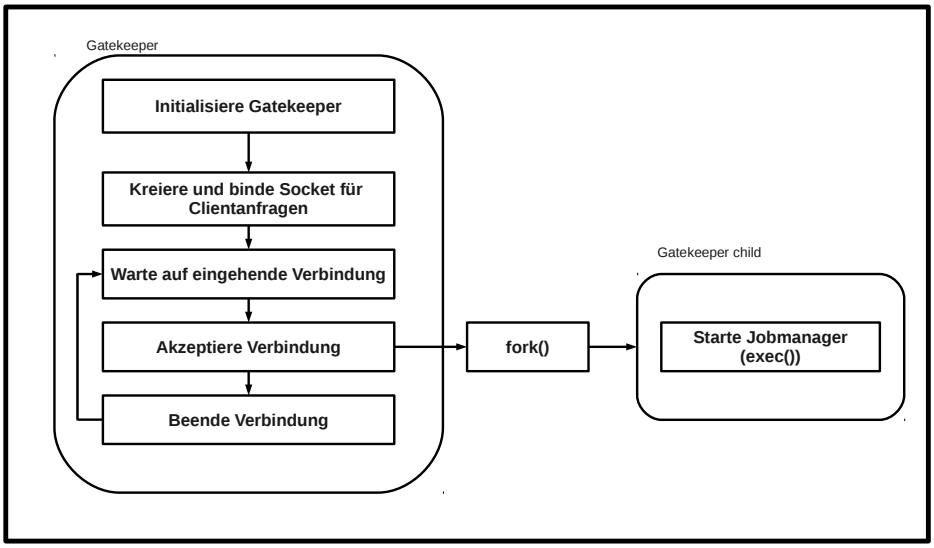

Abbildung 4.3: Gatekeeper als Dämon gestartet.

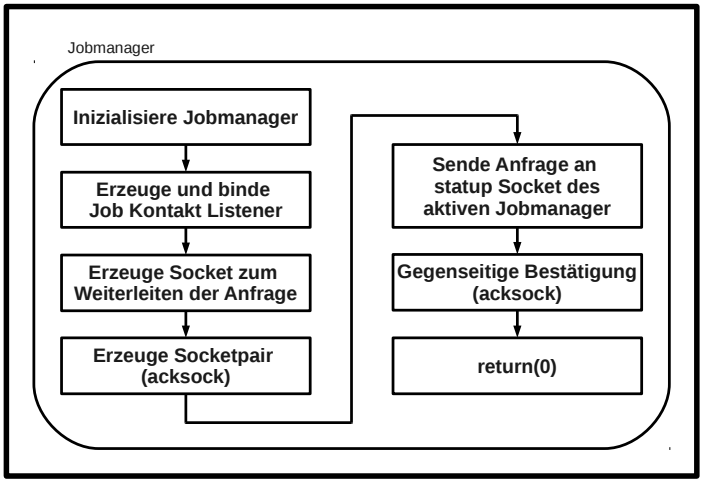

Abbildung 4.4: Jobmanager reicht initiale Anfrage an für User/LRM zuständigen Jobmanager weiter. Kein fork().

in ein dem jeweiligen LRM angepasstes Format umwandelt und den jeweiligen Auftrag an diese übermittelt. Um mit GridTP Servern zu interagieren nutzt das Skript das globus-url-copy Programm. Des weiteren implementiert es Funktionalitäten für Statusabfragen und Jobabbrüche.

Scheduler Event Generator: Der Scheduler Event Generator (SEG) nutzt Scheduler spezifische Module, Scheduler Event Generator LRM Module, um Statusänderungen, die die Local Resource Manager in ihre Logfiles schreiben, auszulesen, in ein Format das die Job Manager versteht umzuwandeln und in eigenen Logfiles zu speichern. Weitere Arbeitsschritte initialisiert er, indem er den Status des Jobs im zuständigen Manager ändert und die Zustandsmaschine startet. Pro Local Resource Manager muss eine Instanz des Scheduler Event Generators laufen, welche die Logfiles regelmäßig nach aktuellen Einträgen durchsucht. Die Instanzen werden in der Regel vor dem Start der anderen Komponenten, über ein Setup Skript, gestartet. Sie laufen unabhängig von den Job Managern, sodass jeweils nur ein Prozess mit dem zugehörigen Locale Resource Manager interagieren muss. Der Scheduler Event Generator ist standardmäßig nicht aktiviert, sondern die Status werden in regelmäigen Abständen über einen Pull-Mechanismus abgefragt.

GRAM Audit Database: Zu Kontrollzwecken können mittels GRAM Informationen wie Startzeit, Stopzeit, etc. von Jobs gespeichert werden. Mit dem globus-gram-audit Programm können diese an eine SQL Datenbank übertragen werden. Gram Audit ist standardmäßig nicht aktiviert.

4.1.2 Client Interaktionen mit den Anwendungsprozessen

GRAM Protokoll: Die Kommunikationen zwischen Client, Gatekeeper und Job Manager erfolgen asynchron mit Hilfe des GRAM Protokolls in Version 2. Es baut auf einer Untermenge des HTTP/1.1 Protokolls [41] auf und versendet GRAM Nachrichten als HTTP POST Nachrichten. Die eigentlichen Anfragen und Antworten befinden sich im Body der Nachricht. Eine Typisch Jobanfrage ist dabei wie folgt aufgebaut:

POST job-manager-name[@user-name] HTTP/1.1
Host: host-name
Content-Type: application/x-globus-gram
Content-Length: message-size

protocol-version: version
job-state-mask: mask
callback-url: callback-contact
rsl: rsl-description

Die Header Variablen beinhalten:

- job-manager-name:
  Der Name des angefragten GRID Service in der Form Jobmanager-LRM.
- user-name:
  Optionale Angabe eines Usernamens, unter dem der Service laufen soll.

- host-name:
  Der Name des Host auf dem der Gatekeeper läuft

- message-size:
  Größe der Nachricht exklusive des Headers.

Die Variablen im Boddy:

- version:
  GRAM Protokoll Version. Also: „2“

- job-state-mask:
  Eine Integer Darstellung der Statusänderungen, über die der Client informiert werden möchte.

- callback-contact:
  Die https URL des Listeners an den die Statusupdates gesendet werden sollen.

- rsl-description:
  Die Beschreibung der Jobanforderungen mittels der Resource Spezifikation Language.

Resouce Spezifikation Language: Für Jobanfragen werden die jeweiligen Anforderungen mittels der Resouce Spezifikation Language Version 1.1 (RSL v1.1) formuliert. Sie ist eine erweiterbare Austauschsprache, die auf ATTRIBUT, VALUE Paaren basiert, welche sich zu nicht trivialen Beschreibungen zusammensetzen lassen. Die Attribute dienen als Parameter um das Verhalten einer oder mehreren Komponenten zu beeinflussen. Die jeweiligen Komponenten müssen dabei nicht alle Attribute verstehen, sondern sie verarbeiten die ihnen bekannten und reichen die unbekannten Attribute unverändert weiter.
Hier einige Attribute als Beispiele:

- arguments:
  Die Kommandozeilen Argumente für die auszuführende Datei.

- executable:
  Der Name der ausführbaren Datei, die auf der Ressource gestartet werden soll.

- two_phase:
  Zum Benutzen von 2-Phasen-Commits vor dem Start und am Ende eines Jobs.

- username:
  Um sicherzustellen, dass ein Job unter einem bestimmten Usernamen läuft.

Client - Gatekeeper: Die initiale Jobanfrage und die Absetzung eines Pings sind die einzigen Anfragen von Clients, die an den Gatekeeper gestellt werden. Die weitere Kommunikation erfolgt, nach erfolgreicher Initialisierung des Jobs, direkt mit dem zuständigen Job Manager. Wird der Gatekeeper über einen Superserver gestartet, so läuft die Kommunikation mit dem Client über den vom Superserver bereitgestellten Socket. Läuft er als Dämon so wird mittels der Funktion net_setup_listener() ein eigener Socket erstellt, der die Anfragen entgegen nimmt.

Client - Job Manager: Der Job Manager erzeugt einen Job-Contact-Listener um eingehende Nachrichten zu bearbeiten. Anfragen werden dabei an den Job-Contact, also die URL des Listeners, gesendet. Der Job-Contact wird dem Client in der Antwort auf die initiale Jobanfrage mitgeteilt.
Für Anfragen an den Job Manager ist der Aufbau der GRAM Protokoll Nachrichten für folgende Aufgaben spezifiziert:

- Statusanfragen
- An- und Abmelden von Callback-Contacts, also Clients die über Statusänderungen informiert werden sollen.
- Abbruch des Jobs.
- Senden eines Signals an den Job.
- Zum generieren eines neuen Proxy Zertifikats.

Um Anfragen zu bearbeiten, ruft der Job-Contact-Listener eine Callbackfunktion auf, welche den Status des Jobs ändert und gegebenenfalls die Zustandsmaschine aktiviert. Nur bei Statusabfragen (Status Pull) entnimmt die Funktion den aktuellen Status aus einer internen Tabelle und teilt diesen den registrierten Clients mit.
Der Job Manager sendet bei Statusänderungen Jobstatus Nachrichten an alle registrierten Clients (Callback-Contacts). Über welche Änderungen ein Client informiert werden will, kann er über die Job State Mask angeben. Für eine zuverlässige Jobübermittlung bietet GRAM5 optional an, ein zwei Phasen Commit durchzuführen. Damit kann sichergestellt werden, dass der Client den Job-Contact für Anfragen kennt. Auch bei der Beendigung von Jobs kann ein zwei Phasen Commit zum Einsatz kommen. Der Job Manager wartet dann mit dem Aufräumen des Systems bis das Commit erfolgt ist.

4.1.3 Interaktionen der Prozesse untereinander

Gatekeeper - Job Manager: Der Gatekeeper startet den Job Manager durch den exec() [42] Befehl. Dabei übergibt er beim ersetzen des Programmcodes im Speicher, durch den Code des Job Managers, auch die Umgebungsvariablen an diesen und leitet so Informationen an ihn weiter. Er kann auch derart konfiguriert werden, dass er einen Kindprozess für den Start erzeugt und der Vaterprozess auf die Beendigung des Kindes wartet. In diesem Fall wird die korrekte Erzeugung des Kindprozesses mittels dafür erstellter Pipes überprüft.

langlebiger Job Manager - kurzlebiger Job Manager Bei der Initiierung der langlebigen Job Manager erzeugen diese mittels globus_gram_job_manager_startup_socket_init() Sockets, die Startup Sockets, um Jobanfragen von anderen Job Managern entgegennehmen zu können. Über die Funktion globus_gram_job_manager_starter_send() erzeugen die kurzlebigen Instanzen Sockets zum übermitteln der Anfragen an die Startup Sockets der jeweils zuständigen langlebigen Job Manager und zusätzlich ein Socketpaar für die Bestätigung der Übermittlung, die so genannten Acksocks. In den Abbildungen 4.4 und 4.2 wird dies noch einmal graphisch dargestellt. Wie in Abbildung 4.5 zu sehen, wird beim empfangen einer Anfrage über den Startup Socket eine Callback Funktion aufgerufen. Diese nimmt die Nachricht entgegen,

bestätigt den Empfang über die Achsocks und startet die Zustandsmaschine. Hauptinhalt der Übertragung sind die Filedeskriptoren des http_body_files und des Security Kontexts, der Deskriptor über den die Antworten versendet werden und einer der angesprochenen Acksocks.

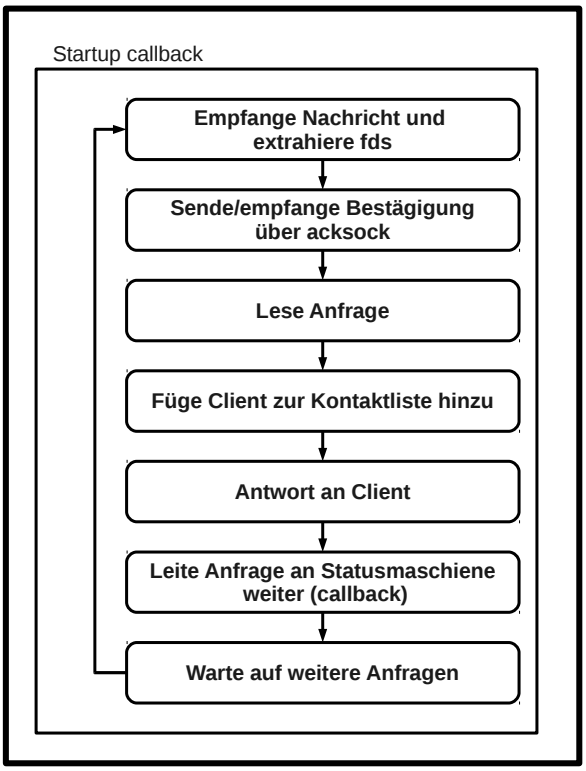

Abbildung 4.5: Startup Socket Callback.

Job Manager - Job Manager Script: Um Anfragen an den jeweiligen Local Resource Manager weiterzuleiten oder einen Filetransfer einzuleiten wird das Globus Job Manager Script genutzt. Für Anfrage wird dazu die Jobbeschreibung, für diesen Zweck erstellte Umgebungsvariablen und das jeweils auszuführende Kommando, über die von XIO zu Verfügung gestellte Funktion popen(), an das Skript übermittelt. Anhand des übermittelten Kommandos entscheidet das Skript, ob im weiteren Verlauf ein Filetransfer, eine Übermittlung des Jobs an den Local Resource Manager, eine Statusabfrage oder ein Jobabbruch erfolgen soll. Als Rückgabewert sendet das Job Manager Skript einen Job Identifier, den angefragten Status oder einen Fehlercode.

Scheduler Event Generator - Job Manager Über Statusänderungen, die durch die Abarbeitung der Jobs entstehen und die die Local Resource Manager in ihren Logfiles vermerken, informiert der Scheduler Event Generator die Job Manager, indem er den Status ändert und die Status Maschine aktiviert. Diese entscheidet über das weitere Vorgehen. Falls der Scheduler Event Generator nicht genutzt werden soll, erfolgt eine periodische Statusabfrage über die Pull Funktionalität des Job Manager Scripts.

4.1.4 Interaktionen mit externen Anwendungen

Job Manager Script - Local Resource Manager: Um ausführbare Dateien oder andere Files von oder zu einem externen Server zu übertragen, nutzt das Globus Job Manager Script das, von GridFTP zu Verfügung gestellte, globus-url-copy Programm. Soll ein Job an einen Local Resource Manager (LRM) zur Ausführung übergeben werden, so wird mit Hilfe des zuständigen Job Manager LRM Adapters ein ausführbares Skript, welches die auszuführende Datei bzw. Dateien und die zugehörigen Parameter beinhaltet, erstellt. Dieses wird anschließend durch eine von den jeweiligen LRM bereitgestellte Subroutine ausgeführt und diese kümmern sich nun um die weitere Bearbeitung. Die jeweiligen Skripte halten zudem Funktionen für Statusabfragen und den Abbruch von Jobs bereit. Diese nutzen wieder Methoden welche von den LRM bereitgestellt werden.

Scheduler Event Generator - Local Resource Manager: Der Scheduler Event Generator durchsucht regelmäßig die Logfiles der Local Resouce Manager. Falls er einen Statuswechsel entdeckt, ändert er den Status des Jobs im Job Manager und aktiviert die Zustandsmaschine.

4.2 Identifizieren der Ressourcen

Hier werden die von den Komponenten genutzten Hauptressourcen vorgestellt. Abbildung 4.6 veranschaulicht diese im Kontext einer Jobübermittlung, bei bereits bestehendem langlebigen Manager, inklusive eventuell vorkommenden weiteren Anfragen und unter Einsatz des Scheduler Event Generators.

Gatekeeper: Die folgenden Ausschnitte stammen aus der Datei gt5.0.2-all-source-installer/source-trees/gatekeeper/source/globus_gatekeeper.c. Die Variable gatekeeperhome beinhaltet, sofern sie nicht mit beim Start des Gatetkeepers über einen Parameter überschrieben wird, entweder das aktuelle Arbeitsverzeichnis, falls der Gatekeeper als Dämon läuft, oder „/etc“, falls er über einen Superserver gestartet wird. Hier die entsprechenden Zeilen im Quellcode:

```
00186 #ifndef GLOBUS_GATEKEEPER_HOME
00187 #       define GLOBUS_GATEKEEPER_HOME "/etc"
  ...
00598     if (getpeername(0, (struct sockaddr *) &name, &namelen) < 0)
00599     {
00600         /* set current working directory as default for
00601          * gatekeeperhome (-home path) when not run from inetd.
00602          * otherwise it is NULL
00603          */
00604 #if defined(TARGET_ARCH_LINUX) || defined(TARGET_ARCH_SOLARIS)
00605         /*
00606          * There is a memory corruption bug in the getcwd in
00607          * glibc-2.1.1 and earlier
00608          *
00609          * Solaris 2.5.1 does not have a correct implementation
00610          * of getcwd either.
00611          *
00612          */
```

```
00613         {
00614             char tmppath[PATH_MAX];
00615
00616             if(getwd(tmppath))
00617             {
00618                 gatekeeperhome = strdup(tmppath);
00619             }
00620         }
00621 #else
00622         {
00623             char *tmppath = NULL;
00624             int size = 1;
00625             while ( (tmppath = getcwd (NULL, size)) == NULL )
00626             {
00627                 size++;
00628             }
00629             gatekeeperhome = tmppath;
00630         }
00631 #endif
00632         run_from_inetd = 0;
00633     }
00634     else
00635     {
00636         run_from_inetd = 1;
00637         gatekeeperhome = GLOBUS_GATEKEEPER_HOME;
...
00744         else if ((strcmp(argv[i], "-home") == 0)
00745                  && (i + 1 < argc))
00746         {
00747             /* Also known as the ${deploy_prefix} */
00748             gatekeeperhome = argv[i+1];
00749             i++;
```

In der Datei gatekeeperhome/globus-gatekeeper.conf können Konfigurationen des Gatekeepers gespeichert und beim Start an diesen übergeben werden. Der Ort kann aber beliebig gewählt werden, da er dem Gatekeeper beim Start mitgeteilt wird. Obwohl mit der Datei das weitere Verhalten maßgeblich beeinflußt werden kann, stellt sie kein wertvolles Ziel für einen Angreifer dar, weil auf sie nur lesend zugegriffen wird (vergleiche Zeile 682) und für ihre Manipulation Administrationsrechte benötigt werden.

```
00670     if ( argc == 3 &&
00671          (!strcmp(argv[1],"-c") ||
00672           !strcmp(argv[1],"-conf"))
00673         )
00674     {
00675         char ** newargv;
00676         char * newbuf;
00677         int newargc = 52;
00678         int  pfd;
00679
00680         newargv = (char**) malloc(newargc * sizeof(char *)); /* not freeded */
00681         newbuf = (char *) malloc(BUFSIZ);  /* dont free */
```

```
00682         newargv[0] = argv[0];
00683         pfd = open(argv[2],O_RDONLY);
```

Eine der zentralen Ressourcen des Gatekeepers ist der über den Socket empfangene Jobauftrag im Gram Protokoll Format. Die Nachricht wird mit Hilfe der GSSAPI entpackt, der angefragte Service aus dem Header extrahiert und die Existenz des zum angefragten Service gehörenden Konfigurationsfiles, standardmäßig unter gatekeeper/grid-services/service_name (vergleiche Zeile 1765), überprüft.

```
\begin{small}
00248 static char *   grid_services = "etc/grid-services";
...
00757         else if ((strcmp(argv[i], "-grid_services") == 0)
00758                  && (i + 1 < argc))
00759         {
00760             grid_services = argv[i+1];
00761             i++;
00762         }
...
01765                 genfilename(gatekeeperhome,grid_services,service_name),
```

Ein Teil der Nachricht, der Http Body, der die Jobbeschreibungen in der Resource Spezification Language beinhaltet, wird im http_body_file, im aktuellen Arbeitsverzeichnis, gespeichert. Der zugehörigen Filedeskriptor wird als Wert der Umgebungsvariable GRID_SECURITY_HTTP_BODY_FD gesetzt (vergleiche Zeile 1691) und somit dem Start des Job Managers an diesen übergeben. Der Inhalt der Nachricht, insbesondere der des http_body_files, findet weitreichend Einzug in den weiteren Ablauf und sie bildet den zentralen Einstiegspunkt für Angreifer.

```
01339     FILE *                              http_body_file;
...
01681         if (! got_ping_request)
01682         {
01683             http_body_file = tmpfile();
01684             if (http_body_file)
01685             {
01686                 size_t body_length;
01687
01688                 setbuf(http_body_file,NULL);
01689                 fcntl(fileno(http_body_file), F_SETFD, 0);
01690                 sprintf(buf, "%d", fileno(http_body_file));
01691                 setenv("GRID_SECURITY_HTTP_BODY_FD", buf, 1);
01692                 notice2(0,"GRID_SECURITY_HTTP_BODY_FD=%s",buf);
```

Im context_tempfile wird der Security Context Token, der bei der Authentifizierung von der GSSAPI erstellt wird, gespeichert. Der Filedeskriptor, mit dessen Hilfe den auf den Inhalt des Tokens zugegriffen werden kann, wird als Wert der Umgebungsvariablen GRID_SECURITY_CONTEXT_BODY_FD (vergleiche Zeile 2159) gesetzt. So kann der aufgebaute Sicherheitskontext für die weitere Kommunikation mit dem Client, im Job Manager, verwendet werden.

```
01350     FILE *                              context_tmpfile = NULL;
...
02147      * export the security context which will destroy it.
02148      * This will also destroy the ability to wrap any error
02149      * messages, so we do this very late.
02150      * First we get an temp file, open it, and delete it.
02151      */
02152
02153     context_tmpfile = tmpfile();
02154     if (context_tmpfile)
02155     {
02156         setbuf(context_tmpfile,NULL);
02157         fcntl(fileno(context_tmpfile), F_SETFD, 0);
02158         sprintf(buf, "%d", fileno(context_tmpfile));
02159         setenv("GRID_SECURITY_CONTEXT_FD", buf, 1);
02160         notice2(0,"GRID_SECURITY_CONTEXT_FD=%s",buf);
```

Beeinflusst wird der Aufbau des Sicherheitskontexts zum einen von den Dateien im Verzeichnis gatekeeperhome/x509_cert_dir, welche die akzeptierten Zertifizierungstellen und eine Beschreibung des geforderten Aufbaus der Zertifikate enthalten, und zum anderen von dem übermittelten Client Zertifikat, das in der Datei gatekeeperhome/x509_user_cert gespeichert wird.

Das zum Aufbau einer wechselseitig authentifizierten Verbindung benötigte Service Zertifikat wird vom Gatekeeper mit Hilfe der GSSAPI angefordert und in der credential_handle Variablen gespeichert.

```
00817         else if ((strcmp(argv[i], "-x509_cert_dir") == 0)
00818                  && (i + 1 < argc))
00819         {
00820             x509_cert_dir = argv[i+1];
00821             i++;
00822         }
...
00969     if (x509_cert_dir)
00970     {
00971         setenv("X509_CERT_DIR",
00972                genfilename(gatekeeperhome,x509_cert_dir,NULL),
00973                1);
```

```
00835         else if ((strcmp(argv[i], "-x509_user_cert") == 0)
00836                  && (i + 1 < argc))
00837         {
00838             x509_user_cert = argv[i+1];
00839             i++;
00840         }
...
00988     if (x509_user_cert)
```

```
00989     {
00990         setenv("X509_USER_CERT",
00991                genfilename(gatekeeperhome,x509_user_cert,NULL),
00992                1);
```

Die Umgebungsvariablen X509_CERT_DIR und X509_USER_CERT werden vor dem Start des Job Managers gelöscht. Es werden Administrationsrechte zum manipulieren der Dateien im x509_cert_dir benötigt und der Zugriff erfolgt nur über die GSSAPI. Sie ist daher für einen Angreifer kein interessantes Ziel. Da sich Nutzerzertifikate entweder schon im Besitz des Angreifers befinden oder eventuell erlangte fremde Zertifikate für einen Angreifer ohne den zugehörigen privaten Schlüssel wertlos sind.

Das vom Client beim Authentifizierungsprozess delegierte Zertifikat und der dazugehörenden private Schlüssel werden unter gatekeeperhome/x509_user_proxy und gatekeeperhome/x509_user_key gespeichert und dem Job Manager mit den Umgebungsvariablen X509_USER_PROXY und X509_USER_KEY übergeben. Sollte ein Angreifer Zugang zu ihnen erhalten, könnte er sie beispielsweise nutzen um auf Servern gespeicherte sensible Daten über GridFTP zu erlangen. Eine initiale Jobanfrage kann er damit aber nicht absetzen.

```
00829         else if ((strcmp(argv[i], "-x509_user_proxy") == 0)
00830                  && (i + 1 < argc))
00831         {
00832             x509_user_proxy = argv[i+1];
00833             i++;
00834         }
...
00981     if (x509_user_proxy)
00982     {
00983         setenv("X509_USER_PROXY",
00984                genfilename(gatekeeperhome,x509_user_proxy,NULL),
00985                1);
00986     }
```

```
00841         else if ((strcmp(argv[i], "-x509_user_key") == 0)
00842                  && (i + 1 < argc))
00843         {
00844             x509_user_key = argv[i+1];
00845             i++;
00846         }
...
00993     }
00994     if (x509_user_key)
00995     {
00996         setenv("X509_USER_KEY",
00997                genfilename(gatekeeperhome,x509_user_key,NULL),
00998                1);
00999     }
```

Um zu prüfen, ob der Client für die Nutzung des angefragten Service autorisiert ist und zum ermitteln des UNIX Usernamen unter welchem der Job Manager laufen soll, wird mittels GSSAPI auf das Gridmapfile, gatekeeperhome/grid-security/grid-mapfile, zugegriffen. Zwar stellt es für einen Angreifer ein wertvolles Ziel dar, aber außer über die GSSAPI besteht im Gatekeeper keine Möglichkeit darauf zuzugreifen. Zudem werden zur manipulation Rootrechte benötigt.

Die ausgeführten oder gescheiterten Aktionen des Gatekeepers werden, falls dies nicht beim Start über die Komandozeilenargumente oder die Konfigurationsdatei deaktiviert wurde, in einer Logdatei, standardmäßig unter gatekeeperhome/logfile, gespeichert. Falls der Gatekeeper als Dämon gestartet wird, wird beim akzeptieren von neuen Verbindungen die alte Logdatei umbenannt und eine neue Datei benutzt (vergleiche ab Zeile 2375). Sollte die Möglichkeit bestehen, gezielt Einträge zu fälschen, bestehende zu löschen oder ganze Datei zu entfernen, so könnten damit Aktionen von Angreifern verschleiert werden. Daher stellen sie ein beliebtes Angriffsziel dar.

```
00188 #endif
00189 #ifndef LOGFILE
00190 #define LOGFILE ""
00191 #endif
00192
...
00227 static char *   logfile = LOGFILE;
...
00731         else if (((strcmp(argv[i], "-l") == 0) ||
00732                   (strcmp(argv[i], "-logfile") == 0))
00733                  && (i + 1 < argc))
00734         {
00735             logfile =  argv[i+1];
00736             i++;
00737         }
...
02375         if (logrotate)
02376         {
02377             time_t clock = time((time_t *) 0);
02378             struct tm *tmp = localtime(&clock);
02379             char buf[128];
02380
02381             sprintf(buf, "logfile rotating at %04d-%02d-%02d %02d:%02d:%02d",
02382                 tmp->tm_year + 1900, tmp->tm_mon + 1, tmp->tm_mday,
02383                 tmp->tm_hour, tmp->tm_min, tmp->tm_sec);
02384
02385             notice2(LOG_INFO, "%s", buf);
02386
02387             if (logging_usrlog)
02388             {
02389                 static int seqnr;
02390                 char *logpath = genfilename(gatekeeperhome, logfile, NULL);
02391                 char *oldpath = malloc(strlen(logpath) + 64);
02392
02393                 sprintf(oldpath, "%s.%04d%02d%02d%02d%02d%02d.%d", logpath,
02394                     tmp->tm_year + 1900, tmp->tm_mon + 1, tmp->tm_mday,
```

```
02395                     tmp->tm_hour, tmp->tm_min, tmp->tm_sec, seqnr++);
...
```

Sollen Informationen für Abrechnungszwecke gespeichert werden, kann ein Accountingfile angegeben werden. Dies liegt dann für gewöhnlich unter globuslocation/acctfile. Ist keines angegeben, so werden die Informationen in die Logdatei geschrieben. Auch hier wird bei jeder neuen Verbindung das alte File gegen ein neues ersetzt. Auch diese Dateien kommen für einen Angriff in Frage. Gelöschte oder gefälschte Einträge könnten einen erheblichen finanziellen Schaden anrichten. Dafür werden aber mindestens die Rechte des Besitzers der Datei benötigt (vergleiche Zeile 450).

```
00738         else if ((strcmp(argv[i], "-acctfile") == 0)
00739                  && (i + 1 < argc))
00740         {
00741             acctfile = argv[i+1];
00742             i++;
00743         }
...
00392 /*************************************************************************
00393 Function:       new_acct_file()
00394 Description:    Rotate old and open new job accounting file.
00395 Parameters:
00396 Returns:
00397 *************************************************************************/
00398 static void
00399 new_acct_file(void)
00400 {
00401     static int acct_fd = -1;
00402
00403     if (acct_fd >= 0)
00404     {
00405         if (strcmp(acctfile, logfile) != 0)
00406         {
00407             static int seqnr;
00408             char *acctpath = genfilename(gatekeeperhome, acctfile, NULL);
00409             char *oldpath = malloc(strlen(acctpath) + 64);
00410             time_t clock = time((time_t *) 0);
00411             struct tm *tmp = localtime(&clock);
00412             int ret;
00413
00414             sprintf(oldpath, "%s.%04d%02d%02d%02d%02d%02d.%d", acctpath,
00415                 tmp->tm_year + 1900, tmp->tm_mon + 1, tmp->tm_mday,
00416                 tmp->tm_hour, tmp->tm_min, tmp->tm_sec, seqnr++);
00417
00418             if ((ret = rename(acctpath, oldpath)) != 0)
00419             {
00420                 notice4(LOG_ERR, "ERROR: cannot rename %s to %s: %s",
00421                     acctpath, oldpath, strerror(errno));
00422             }
00423             else
00424             {
00425                 notice2(0, "renamed accounting file %s", oldpath);
```

```
00426             }
00427
00428             free(acctpath);
00429             free(oldpath);
00430
00431             if (ret) {
00432                 return;
00433             }
00434         }
00435
00436         close(acct_fd);
00437         acct_fd = -1;
00438     }
00439
00440     if (!acctfile)
00441     {
00442         acctfile = logfile;
00443     }
00444
00445     if (acctfile && *acctfile)
00446     {
00447         const char *acct_fd_var = "GATEKEEPER_ACCT_FD";
00448         char *acctpath = genfilename(gatekeeperhome, acctfile, NULL);
00449
00450         acct_fd = open(acctpath, O_WRONLY | O_APPEND | O_CREAT, 0644);
00451
00452         if (acct_fd < 0)
00453         {
00454             notice3(LOG_ERR, "ERROR: cannot open accounting file '%s': %s",
00455                 acctpath, strerror(errno));
00456
00457             unsetenv(acct_fd_var);
00458         }
00459         else
00460         {
00461             /*
00462              * Now inform JM via environment.
00463              */
00464
00465             char buf[32];
00466
00467             sprintf(buf, "%d", acct_fd);
00468
00469             setenv(acct_fd_var, buf, 1);
00470
00471             notice4(0, "%s=%s (%s)", acct_fd_var, buf, acctpath);
00472         }
00473
00474         free(acctpath);
00475     }
00476 }
```

Job Manager: Die Variabel GLOBUS_LOCATION entspricht der Variablen gatekeeperhome, falls sie nicht explizit beim Start des Job Managers geändert wird. Die Konfiguration des Job Managers ist standardmäßig in der Datei GLOBUS_LOCATION/globus-job-manager.conf gespeichert. Unter GLOBUS_LOCATION/etc/grid-services/jobmanager-LRM sind die Local Resource Manager spezifischen Konfigurationdateien zu finden. Auf den Http Body und den Security Kontext wird mittels der Filedeskriptoren, welche den vom Gatekeeper übergebenen Umgebungsvariablen entnommen werden, zugegriffen.

Das Arbeitsverzeichnis wird beim Aufruf der Funktion globus_gram_job_manager_init() in Zeile 211 der Datei main(), unter gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source/, erstellt. Durch die beim Erzeugen des Ordners angegebene Option S_IRWXU werden nur dem Erzeuger Lese-, Schreibe- und Ausführungsrechte zugestanden. Die Funktion globus_gram_job_manager_init() befindet sich in der Datei globus_gram_job_manager.c im selben Pfad.

```
00203     dir_prefix = globus_common_create_string(
00204             "%s/.globus/job/%s",
00205             manager->config->home,
00206             manager->config->hostname);
00207     if (dir_prefix == NULL)
00208     {
00209         rc = GLOBUS_GRAM_PROTOCOL_ERROR_MALLOC_FAILED;
00210         goto malloc_dir_prefix_failed;
00211     }
00212     rc = globus_l_gram_mkdir(dir_prefix);
00213     if (rc != GLOBUS_SUCCESS)
00214     {
00215         goto mkdir_failed;
00216     }
```

Um zu signalisieren, dass ein Job Manager für eine User/LRM Kombination zuständig ist, wird eine Sperre auf das Lockfile, GLOBUS_LOCATION/jobmanager_type/service_tag.lock, gesetzt. Nur wenn diese Sperre gesetzt werden kann, wird im weiteren Verlauf ein Startup Socket erzeugt und der Job Manager wird zur langlebigen Instanz, die zuständig für alle Aufträge dieser User/LRM Kombination ist. Falls ein Angreifer in der Lage ist, Lockfiles ander Nutzer zu manipulieren, so könnte er den Service empfindlich stören. Dazu ist allerdings mindestens eine horizontale Rechteausweitung vonnöten.

gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source/
globus_gram_job_manager.c:

```
00279     manager->lock_path = globus_common_create_string(
00280             "%s/%s.%s.lock",
00281             dir_prefix,
00282             manager->config->jobmanager_type,
00283             manager->config->service_tag);
```

Der Socket Dateipfad für den Startup Socket des zuständigen Job Managers ist GLOBUS_LOCATION/jobmanager_type/service_tag.socket. Da bei der Erzeugung Schreibrechte nur an den Besitzer und die zugehörige Gruppe vergeben werden (umask(S_IRWXG|S_IRWXO)),

können Angreifer keine Aufträge einschleusen und dieser Socket stellt keinen besonderen Wert für sie dar.

gt5.0.2-all-source-installer/source-trees/gram/jobmanager/
source/globus_gram_job_manager.c:

```
00290     manager->socket_path = globus_common_create_string(
00291             "%s/%s.%s.sock",
00292             dir_prefix,
00293             manager->config->jobmanager_type,
00294             manager->config->service_tag);
```

Um Anfragen zu laufenden Jobs entgegen zunehmen, wird ein TCP-Listener, der Job Contact Listener, erstellt. An diesen können Clients im Gram Protokoll erstellte Nachrichten senden. Die URL, im Format https://host:port/, wird ihnen als Contact String mitgeteilt.

Das benutzte Proxyzertifikat wird unter GLOBUS_LOCATION/jobmanager_type/service_tag.cred gespeichert und die Grid Security Infrastructure wird angewiesen, dieses zukünftig beim Übermitteln von Jobstate Nachrichten und zur Kommunikation mit GridFTP Servern zu verwenden.

gt5.0.2-all-source-installer/source-trees/gram/jobmanager/
source/globus_gram_job_manager.c:

```
00218     manager->cred_path = globus_common_create_string(
00219             "%s/%s.%s.cred",
00220             dir_prefix,
00221             manager->config->jobmanager_type,
00222             manager->config->service_tag);
```

Die mit neuen, an den Job Contact Listenener gestellten, Anfragen ankommenden Proxyzertifikate werden unter GLOBUS_LOCATION/.globus/job/HOSTNAME/JOB_ID/x509_user_proxy abgelegt und ersetzen anschließend die aktuell zur Kommunikation genutze Versionen.

gt5.0.2-all-source-installer/source-trees/gram/jobmanager/
source/globus_gram_job_manager_request.c:

```
03370 globus_l_gram_export_cred(
03371     globus_gram_jobmanager_request_t *  request,
03372     gss_cred_id_t                       cred,
03373     const char *                        job_directory,
03374     char **                             proxy_filename)
03375 {
...
03414     filename = globus_common_create_string(
```

```
03415             "%s/x509_user_proxy",
03416             job_directory);
```

Die Status der einzelnen Jobs werden in den Dateien GLOBUS_LOCATION/tmp/gram_job_state/logname.hostname.uniq_id (vergleiche Zeile 49) gespeichert. Um beim schreiben race conditions zu vermeiden, so wird das zugehörige state_lock_file (vergleiche Zeile 74) verwendet. Wird ein langlebiger Job Manager neu gestartet, wird das gram_job_state Verzeichnis nach existierenden Dateien durchsucht, um die damit gefundenen Jobs neu zu laden.

gt5.0.2-all-source-installer/source-trees/gram/jobmanager/
source/globus_gram_job_manager_state_file.c:

```
00041 int
00042 globus_gram_job_manager_state_file_set(
00043     globus_gram_jobmanager_request_t *  request,
00044     char **                             state_file,
00045     char **                             state_lock_file)
00046 {
00047     int                                 rc = GLOBUS_SUCCESS;
00048
00049     if(request->config->job_state_file_dir == GLOBUS_NULL)
00050     {
00051         *state_file = globus_common_create_string(
00052                 "%s/tmp/gram_job_state/%s.%s.%s",
00053                 request->config->globus_location,
00054                 request->config->logname,
00055                 request->config->hostname,
00056                 request->uniq_id);
00057     }
00058     else
00059     {
00060         *state_file = globus_common_create_string(
00061                 "%s/job.%s.%s",
00062                 request->config->job_state_file_dir,
00063                 request->config->hostname,
00064                 request->uniq_id);
00065     }
00066
00067     if (*state_file == NULL)
00068     {
00069         rc = GLOBUS_GRAM_PROTOCOL_ERROR_MALLOC_FAILED;
00070
00071         goto create_state_file_failed;
00072     }
00073
00074     *state_lock_file = globus_common_create_string(
00075             "%s.lock",
00076             *state_file);
```

Zum validieren von Anfragen wird das generische Validationfile, GLOBUS_LOCATION/share/globus_gram_job_manger/globus-gram-job-manager.rvf (Zeile 135), und das jeweilige Local Resource Manager spezifische Validationfile GLOBUS_LOCATION/share/globus_gram_job_manger/jobmanager-type (Zeile 145), benutzt. Mit ihnen kann zwar die Existenz benötigter Einträge überprüft werden, unbekannte Einträge werden jedoch ignoriert und unverändert weitergegeben. Ein Angreifer mit Administrationsrechten könnte die Datei derart verändern, dass alle Anfragen als ungültig abgewiesen werden und so die Verfügbarkeit stören.

gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source/globus_gram_job_manager_validate.c:

```
00123 globus_gram_job_manager_validation_init(
00124     globus_gram_job_manager_t *         manager)
00125 {
00126     char *                              validation_filename;
00127     char *                              scheduler_validation_filename;
00128     int                                 rc = GLOBUS_SUCCESS;
00129     globus_list_t *                     tmp_list;
00130     globus_gram_job_manager_validation_record_t *
00131                                         record;
00132
00133     manager->validation_records = NULL;
00134
00135     validation_filename = globus_common_create_string(
00136             "%s/share/globus_gram_job_manager/%s.rvf",
00137             manager->config->globus_location,
00138             "globus-gram-job-manager");
00139
00140     if(validation_filename == NULL)
00141     {
00142         rc = GLOBUS_GRAM_PROTOCOL_ERROR_MALLOC_FAILED;
00143         goto validation_filename_failed;
00144     }
00145     scheduler_validation_filename = globus_common_create_string(
00146             "%s/share/globus_gram_job_manager/%s.rvf",
00147             manager->config->globus_location,
00148             manager->config->jobmanager_type);
00149     if(scheduler_validation_filename == NULL)
```

Logeinträge und Abrechnungsinformationen werden in die vom Gatekeeper erzeugten Dateien geschrieben.

Job Manager Script: Das Job Manager Script erhält von den Job Managern die angepasste Jobbeschreibung und das Proxyzertifikat. Es erzeugt einen Ordner zum ablegen von, mit globus-url-copy importierten, ausführbaren Dateien und sonstigen Daten, die für die weitere Ausführung benötigt werden. Bei Berechungen in Grid Umgebungen können viele externe Daten benötigt werden. Dennoch sollte vom Administrator der Speicherplatz der einzelnen Nutzer begrenzt werden, da ein Angreifer sonst die gesamten Speicherressourcen des Systems belegen könnte. Falls Local Resource Manager angesprochen oder Daten übertragen werden

sollen, wird mit Hilfe der LRM Adapter Module ein ausführbares Skript erzeugt. Optional können LRM spezifische Validationsdateien, zum überprüfen der RSL Syntax, von den LRM Adapter Modulen genutzt werden.

Scheduler Event Generator: GLOBUS_LOCATION/etc/globus-job-manager-seg.conf enthält die Konfigurationsdatei für den globus-job-manager-event-generator. Die Einträge der Datei bestehen aus den Pfaden zu den LRM-unabhängigen SEG Logfiles. Spezielle Konfigurationsdateien für die jeweiligen Local Resource Manager sind unter GLOBUS_LOCATION/etc/globus-LRM.conf zu finden und diese enthalten die Pfade zu den Logdateien der einzelnen Local Resource Manager.

4.3 Vertrauen und Rechte

Aufgrund der Dynamik der Virtuellen Organisationen, und den dadurch entstehenden häufigen Clientwechsel, ist die Clientseite als nicht sicher einzustufen. Es kann davon ausgegangen werden, dass Zertifikate unrechtmäßig erlangt und beliebige Nachrichten, über den mit Transport Layer Security geschützten Transportweg, gesendet werden.

Die Serviceseite steht unter unserer Kontrolle und ist vor unbefugten Zugriffen geschützt. Abbildung 4.7 zeigt eine generische Version des Gatekeepers mit den jeweils gültigen Benutzerrechten und der Behandlung der Authentifizierung sowie der Autorisierung. Zur Kommunikation mit dem Client wird im Gatekeeper, mit Hilfe der Grid Security Infrastructure, eine wechselseitige Authentifizierung vorgenommen und der so erhalten Security Kontext für den weiteren Nachrichtenaustausch mit dem Client, auch in den anderen Komponenten, genutzt. Für Delegationszwecke wird ein Proxyzertifikat vom Client angefordert und signiert. Es hat eine Gültigkeit von 12 Stunden und wird bei weiteren Anfragen, oder durch eine explizite Anfrage nach Erneuerung, durch ein neues ersetzt. Eine Job Manager kann viele Jobs für einen Nutzer bzw. eine Nutzergruppe verwalten. Dabei wird immer das Proxyzertifikat mit der längsten Laufzeit verwendet. Läuft das verwendete Proxyzertifikat aus, so werden die registrierten Clients benachrichtigt und der Job Manager terminiert. Die einzelnen Jobs laufen unabhängig vom Job Manager weiter. Bei Bedarf kann dieser von den Clients neu gestartet werden, wobei ein neues Proxyzertifikat übermittelt wird.

Der Gatekeeper läuft unter mit den Rechten des Rootaccounts. Nach erfolgreicher Authentifizierung über die Grid Security Infrastruktur wird der Client, sofern er zur Nutzung des angefragten Service autorisiert ist, auf einen lokalen Benutzernamen abgebildet. Für den Abgleich auf den lokalen User und die Autorisation wird die Gridmap Datei, mittels der Grid Security Infrastructure, konsultiert. Nach erfolgreicher Abbildung wechselt der Gatekeeper mit Hilfe der Funktion globus_gatekeeper_util_trans_to_user() in den jeweiligen Userkontext, bevor er den Job Manager startet.

Außer dem Scheduler Event Generator laufen alle weiteren Komponenten mit den Rechten des lokalen Users. Der Scheduler Event Generator muss auf die Logfiles aller Local Resource Manager und auf alle Job Manager Zugriff haben. Seine Rechte umfassen daher die Rechte aller autorisierten Nutzer.

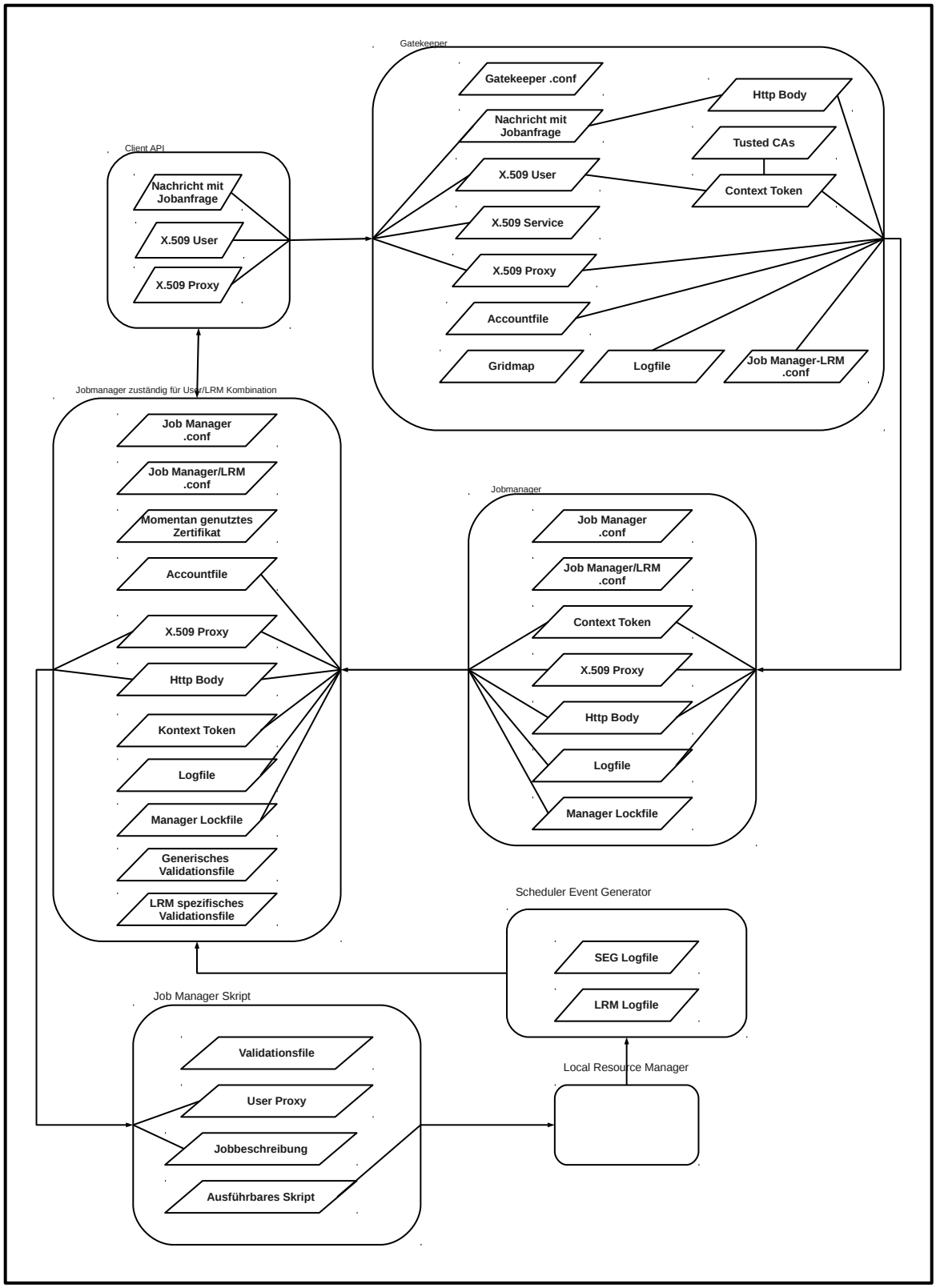

Abbildung 4.6: Die von den Komponenten genutzten Ressourcen. Es existiert bereits ein für die User/LRM Kombination zuständiger Job Manager

Da die Dynamik der Nutzer die Besonderheit von Gridsystemen darstellt, erfolgt die Suche nach Schwachstellen mit Augenmerk auf einen möglichen Angriff über einen Client, der im Besitz eines gültigen Zertifikats ist.

4.4 Auswertung der Komponenten

Mit den in den vorausgegangenen Schritten gewonnenen Erkenntnissen lassen sich nun die Angriffsziele in dem System bestimmen, die für Angreifer die höchsten Werte haben. Diese werden im weiteren Verlauf vorrangig untersucht. Die gesuchten Ziele zeichnen sich dadurch aus, dass sie bei erfolgreicher Kompromittierung eine oder mehrere der folgende Wirkungen haben können:

- Die Verfügbarkeit des Services oder von Daten wird unterbunden.
- Unberechtigte haben Zugriff auf sensible Daten oder können diese verändern.
- Eine unautorisierte Nutzung von Ressourcen wird ermöglicht.

Nachfolgende Punkte beschreiben mögliche Ziele. Sie sind dabei absteigend nach ihrem Wert für einen Angreifer sortiert.

- Das wohl lohnenswerteste Ziel für einen Angreifer scheint der Gatekeeper zu sein. Da er mit Rootrechten läuft, wie in Schritt drei der Auswertung ersichtlich, kann sowohl eine horizontale, wie auch eine vertikale Rechteausweitung möglich sein, falls entsprechende Schwachstellen vorhanden sind. Damit wäre der Zugriff auf fremde und sensible Daten ermöglicht. Es könnten Jobs unter anderen Konten zum laufen gebracht werden oder der Angreifer könnte gar die Kontrolle über das gesamte System erlangen und beliebigen Code mit Rootrechten ausführen. Falls die Erreichbarkeit des Gatekeepers unterbunden wird, können keine neuen Jobs mehr abgesetzt werden und auf laufende kann nur zugegriffen werden, solange der zuständige Job Manager noch läuft. Dies wird bei der Betrachtung der in Schritt eins evaluierten Interaktionen der Komponenten sichtbar.
- Ein weiteres Angriffsziel stellen die Abrechnungsdaten in den Accountfiles dar. Sie könnten zu eigenen Gunsten verändert werden oder es könnten die Abrechnungsdaten anderer Nutzer manipuliert werden.
- Der Job Manager, wie auch das Job Manager Script, laufen mit den Rechten des jeweiligen Users und haben daher ein geringeres Schadpotential. Programme mit User Rechten können ohnehin auf der Ressource ausgeführt werden, falls ein Zertifikat recht- oder unrechtmäßig erlangt wurde. Eine erfolgreiche Denial of Service Attacke gegen eine entsprechende Schwachstelle würde wenig Schaden anrichten, da der Job Manager ohne großen Aufwand wieder über den Gatekeeper gestartet werden kann und Jobs unabhängig von ihm weiterlaufen. Es ist jedoch zu prüfen, ob über den Job Contact Listener ein unbefugter Zugriff auf Jobs möglich ist.
- Die Logdateien könnten von einem Angreifer manipuliert werden um Aktionen zu verschleiern.

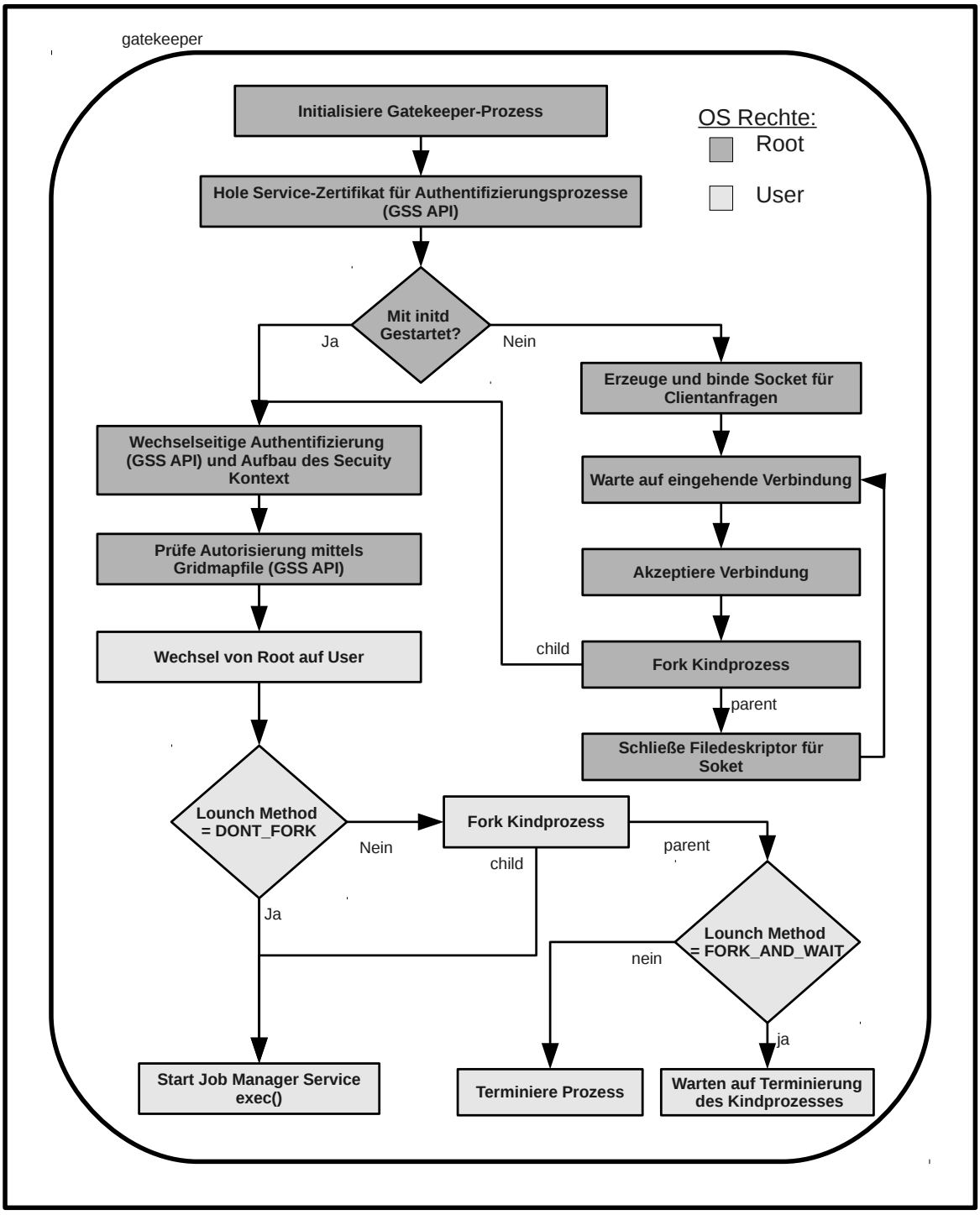

Abbildung 4.7: Generischer Gatekeeper mit Rechten, Authentifizierung und Autorisierung

- Der Scheduler Event Generator läuft mit privilegierten Rechten, Schwachstellen könnten daher eine horizontale Rechteausweitung ermöglichen. Falls eine Lücke es erlaubt Events gezielt zu erzeugen, so könnten beliebige Jobs kontrolliert werden, um beispielsweise die Anzahl der laufenden Jobs zu verringern und dadurch die eigenen schneller zur Ausführung kommen zu lassen.

Auswertung des Gatekeepers

Ein Angriff auf die Verfügbarkeit des Gatekeepers (Denial of Service) ist nur interessant, falls dieser als Dämon läuft. Wird er hingegen mit einem Superserver bei jeder Anfrage neu gestartet, so macht ein Angriff darauf nur Sinn, wenn man dadurch die Last in hohem Maße erhöhen kann. Andere Schwachstellen in diesem Bereich stellen eine geringere Gefahr dar. Läuft er als Dämon, werden auch die Kinderprozesse des Gatekeepers für jede Verbindung neu gestartet, daher sind hier Schwachstellen im Vaterprozess oder Schwächen die es ermöglichen eine hohe Anzahl von Kindprozessen um damit eine Überlast zu erzeugen vonnöten, um die Verfügbarkeit zu gefährden. Da im Fall des Starts als Dämon der Gatekeeper lokal gestartet wird, können die Startpunkte für die Suche nach Lücken daher auf Segmente nach der Initialisierung der Clientkommunikation reduziert werden. Im Bereich der Implementierung wird primär nach Schwächen gesucht, die es ermöglichen durch Eingaben bzw. geschickte Manipulation von Rückgabewerten einzelne Speichersegmente zu überschreiben und so den Gatekeeper zu einem unkontrollierten Verhalten anzuregen. Diese Schwachstellen werden Buffer Overflow Schwachstellen genannt und gehören zum Bereich der Input Validation Schwachstellen. Genauer sind dies hier Stack, Heap und Formatstring Overflow Schwachstellen, auf die der Fokus der Untersuchung gelegt wird. Funktionen die zu Heap Overflow Schwachstellen führen können, kommen aus der malloc() Familie, wie malloc(), realloc() und free(). Beispiele für Funktionen die den Stack betreffen sind gets(), strcpy() und strcat(). Bei der Suche nach Formatstring Schwachstellen hilft die Überprüfung von Funktionen aus der printf() Familie sowie von syslog(). Ausgehend von diesen Funktionen ist zu prüfen, ob Eingaben von Clients genutzt werden oder ob die Eingaben von diesen, wie etwa bei der Verwendung von Rückgabewerten, gezielt beeinflusst werden können und ob diese Eingaben ausreichend validiert werden. Bei der Berechnung der Speicherbereiche unterlaufen Programmierern leicht so genannte Off-By-One Fehler. Das bedeutet, dass sich bei der Berechnung des benötigten Speichers um eins verzählt wird. Die Speichergrößenberechnung ist daher sorgfältig zu prüfen.

Buffer Overflow Schwachstellen im Gatekeeper könnten, falls die Eingaben geschickt gewählt werden, auch dazu führen, dass beliebiger Code mit Rootrechten ausgeführt werden kann. Dadurch könnte eine vertikale Rechteausweitung erfolgen. Im Bereich des Designs ist zu überprüfen ob eine Möglichkeit besteht den Authentifizierungs- oder Autorisierungsprozess zu umgehen. Damit könnte eine horizontale oder vertikale Rechteausweitung erreicht werden. Daher muss geprüft werden ob es möglich ist, den Wechsel von Root auf User ohne Abbruch des Programmflusses zu unterbinden und so die weiteren Komponenten mit Rootrechten laufen zu lassen. In diesem Fall könnten Schwachstellen in sonst unkritischen Bereichen gefährlich werden.

Zusätzlich ist zu überprüfen ob Pfadmanipulationen möglich sind, mit deren Hilfe unbefugter Zugriff auf Dateien erfolgen kann.

Die Quelltexte für den Gatekeeper sind in dem Ordner gt5.0.2-all-source-installer/source-trees/gatekeeper/ zu finden. Falls nicht anders erwähnt, sind die folgenden Ausschnitte aus der Datei globus_gatekeeper.c.

Die Kommunikation zwischen Gatekeeper, der als Dämon läuft, und den Clients startet mit dem akzeptieren der jeweiligen Verbindung in Zeile 1235:

```
01235             connection_fd = net_accept(listener_fd);
```

Ein externer Angreifer kann also mittels Eingabedaten erst ab dort Einfluss nehmen. Hier die ersten Zeilen der Funktion:

```
02321 /*************************************************************************
02322 Function:    net_accept()
02323 Description: Accept a connection on socket skt and return fd of new connection.
02324 Parameters:
02325 Returns:
02326 *************************************************************************/
02327 static int
02328 net_accept(int skt)
02329 {
02330     netlen_t           fromlen;
02331     int                skt2;
02332     int                gotit;
02333     struct sockaddr_in from;
02334
02335     fromlen = sizeof(from);
02336     gotit = 0;
02337
02338     while (!gotit)
02339     {
02340         fd_set         fdset;
02341         struct timeval timeout;
02342         int            n;
02343
02344         FD_ZERO(&fdset);
02345         FD_SET(skt, &fdset);
02346         timeout.tv_sec = 60;
02347         timeout.tv_usec = 0;
02348
02349         n = select(skt + 1, &fdset, (fd_set *) 0, &fdset, &timeout);
02350
02351         if (n < 0 && errno != EINTR)
02352         {
02353             error_check(n, "net_accept select");
02354         }
02355         else if (n > 0)
02356         {
02357             long flags;
02358
02359             skt2 = accept(skt, (struct sockaddr *) &from, &fromlen);
```

Es sind Schwachstellen bekannt, die durch eine fehlende Überprüfung der Größe des fd_sets entstehen und durch die es möglich ist die fd_set Struktur zum überlaufen zu bringen, indem viele Sockets erstellt werden und so die maximale Anzahl der im fd_set verarbeitbaren

Sockets überschritten wird [43]. Dem wurde hier entgegengewirkt, indem beim Erzeugen des Sockets, mittels der Funktion net_setup_listener(int backlog, int *port, int *socket), über die backlog Variable eine maximale Anzahl an gleichzeitig akzeptierten Verbindungen von zwei vorgegeben wurde. Die Variable wird der listen() Funktion in Zeile 2465 übergeben und bewirkt, dass alle weiteren Anfragen zurückgewiesen werden [44]. Zu bemerken ist, das in Zeile 2346 ein Timeout definiert wird, um zu verhindern, dass der Socket blockiert falls Angreifer Verbindungen aufbauen und keine Nachrichten senden.

```
02429 /*************************************************************************
02430 Function:       net_setup_listener()
02431 Description:
02432 Parameters:
02433 Returns:
02434 *************************************************************************/
02435 static void
02436 net_setup_listener(int backlog,
02437                    int * port,
02438                    int * skt)
02439 {
02440     netlen_t        sinlen;
02441     struct sockaddr_in sin;
02442         long flags;
02443     int one=1;
02444
02445     *skt = socket(AF_INET, SOCK_STREAM, 0);
02446     error_check(*skt,"net_setup_anon_listener socket");
02447
02448         flags = fcntl(*skt, F_GETFL, 0);
02449         flags |= O_NONBLOCK;
02450         fcntl(*skt, F_SETFL, flags);
02451
02452     error_check(setsockopt(*skt, SOL_SOCKET, SO_REUSEADDR, (char *)&one, sizeof(one)),
02453                 "net_setup_anon_listener setsockopt");
02454
02455     sin.sin_family = AF_INET;
02456     sin.sin_addr.s_addr = INADDR_ANY;
02457     sin.sin_port = htons(*port);
02458
02459     sinlen = sizeof(sin);
02460
02461     error_check(bind(*skt,(struct sockaddr *) &sin,sizeof(sin)),
02462                 "net_setup_anon_listener bind");
02463
02464
02465     error_check(listen(*skt, backlog), "net_setup_anon_listener listen");
02466
02467     getsockname(*skt, (struct sockaddr *) &sin, &sinlen);
02468     *port = ntohs(sin.sin_port);
02469 }
```

Auch bei der weiteren Untersuchung der Funktion net_accept() konnten keine Stellen entdeckt werden, die den Gatekeeper verwundbar machen.

Nach dem Verbindungsaufbau erzeugt der Gatekeeper mittels fork() ein Kindprozess für die Weiterbearbeitung, schließt die Verbindung und wartet auf weitere Anfragen auf dem Socket. Der Kindprozess ermittelt die Peeradresse (peernum), die für Loggingzwecke genutzt wird, und führt die wechselseitige Authentifizierung über die GSS_Assist API durch.

Ermitteln der Peeradresse:

```
01379     if (getpeername(0, (struct sockaddr *) &peer, &peerlen) == 0)
01380     {
01381         if (peer.sin_family == AF_INET)
01382             peernum = inet_ntoa(peer.sin_addr);
01383         else
01384             peernum = "";
01385     }
01386
01387     fdout = fdopen(dup(0),"w"); /* establish an output stream */
01388     setbuf(fdout,NULL);
01389
01390     notice3(LOG_INFO, "Got connection %s at %s", peernum, timestamp());
```

Wechselseitige Authentifizierung mittels der GSS_assist API:

```
01487     major_status = globus_gss_assist_accept_sec_context(
01488         &minor_status,
01489         &context_handle,
01490         credential_handle,
01491         &client_name,
01492         &ret_flags,
01493         NULL,            /* don't need user_to_user */
01494         &token_status,
01495         &delegated_cred_handle,
01496         globus_gss_assist_token_get_fd,
01497         (void *)stdin,
01498         globus_gss_assist_token_send_fd,
01499         (void *)fdout);
01500
01501     if (major_status != GSS_S_COMPLETE)
01502     {
01503         if (logging_usrlog)
01504         {
01505             globus_gss_assist_display_status(usrlog_fp,
01506                                              "GSS authentication failure ",
01507                                              major_status,
01508                                              minor_status,
01509                                              token_status);
01510         }
01511
01512         failure4(FAILED_AUTHENTICATION,
```

```
01513                 "GSS failed Major:%8.8x Minor:%8.8x Token:%8.8x\n",
01514                 major_status,minor_status,token_status);
01515     }
01516
01517     /* now OK to send wrapped error message */
01518     ok_to_send_errmsg = 1;
```

Ein möglicher Angreifer könnte nun Verbindungen aufbauen, eine kurze Nachricht senden und dann sofort wieder schließen. Die Funktion getpeername() würde einen ENOTCONN Fehler zurück liefern [45]. Obwohl die Arbeitsschritte bis zu der Stelle an der erkannt wird, dass keine Verbindung mehr besteht und der Prozess darauhin über die failure4() Funktion abgebrochen wird, nicht sehr lang sind, wäre eine Abfrage des Fehlers durch einen else Zweig von Vorteil und würde unnötige Last verhindern. Auch wird beim Auftreten dieses Fehlers die Variable peernum nicht initialisiert, wodurch beim Loggen über notice3() in Zeile 1390 ein nicht absehbarer Wert der Variable benutzt wird. Schlägt die Authentifizierung mittels globus_gss_assist_accept_sec_context() fehl, so soll mittels failure() der Client informiert und der Prozess beendet werden. Da aber die Variable ok_to_send_errmsg erst danach auf 1 gesetzt wird, kann der angegebene Fall FAILED_AUTHENTICATION nicht erreicht werden und ist daher überflüssig. Der Prozess wird aber beendet. Die fehlende Benachrichtigung stellt zwar keinerlei Gefahr dar sondern verhindert nur, dass unautorisierte Clients Informationen beziehen können. Sie zeigt aber auch das generell kleinere logische Fehler im Code existieren.

Ausschnitt aus der Funktion failure():

```
02609     if (ok_to_send_errmsg)
02610     {
02611         char * response;
02612
02613         switch (failure_type)

    ...

02636         case FAILED_AUTHENTICATION:
02637         default:
02638             response = ("HTTP/1.1 500 Internal Server Error\015\012"
02639                         "Connection: close\015\012"
02640                         "\015\012");
02641             break;
02642         }
02643
02644         /* don't care about errors here */
02645         globus_gss_assist_wrap_send(&minor_status,
02646                                     context_handle,
02647                                     response,
02648                                     strlen(response) + 1,
02649                                     &token_status,
02650                                     globus_gss_assist_token_send_fd,
02651                                     fdout,
02652                                     logging_usrlog?usrlog_fp:NULL);
02653     }
```

Ein erfolgreicher Denial of Service Angriff auf den Kindprozess nach der Authentifizierung hätte keine Auswirkungen mehr auf die generelle Verfügbarkeit des Gatekeepers.

Die Funktion getpeername() wird auch genutzt um zu entscheiden, ob der Gatekeeper durch einen Superserver gestartet wurde oder als Dämon laufen soll. Wird er über einen Superserver gestartet und dann eine Verbindung aufgebaut und sofort wieder geschlossen, so wird die Rückgabe der Funktion als Start des Gatekeepers als Dämon missinterpretiert. Falls dies nicht erkannt wird, könnte eine große Anzahl an Sockets erzeugt werden.

```
00592      * Decide if we are being run from inetd.
00593      * If so, then stdin will be connected to a socket,
00594      * so getpeername() will succeed.
00595      */
00596
00597     namelen = sizeof(name);
00598     if (getpeername(0, (struct sockaddr *) &name, &namelen) < 0)
00599     {

    ...

00632         run_from_inetd = 0;
00633     }
00634     else
00635     {
00636         run_from_inetd = 1;

    ...

00644     }
```

Dieser Fehler wird durch das Binden des mit der Funktion net_setup_listener() erzeugten Sockets, in Zeile 2461, bemerkt. Die Funktion bind() gibt die Fehlermeldung EADDRINUSE zurück, was aussagt, dass die Adresse bereits benutzt wird. Der Prozess wird darauf hin mittels der Funktion error_check() beendet. Die bis dahin verrichtete Arbeit ist überschaubar, vor allem im Vergleich mit der Last die erzeugt wird, wenn mittels unrechtmäßig erlangter Zertifikaten rechenintensive Jobs abgesetzt werden.

Der nächste kritische Punkt ist die Überprüfung ob versucht wird, mit Hilfe von Navigationssymbolen im Service Namen (service_name) nach Service Files außerhalb des Service Ordners zu suchen. Dies geschieht in Zeile 1759:

```
01756     /* Don't allow the client to look for service files outside of the
01757      * service directory
01758      */
01759     if (strchr(service_name, '/') != NULL)
01760     {
01761         failure2(FAILED_SERVICELOOKUP, "Invalid service name %s", service_name);
01762     }
```

Es ist jedoch möglich das überprüfte „Slash„ Zeichen auch anders zu codieren, beispielsweise durch die Hexadezimaldarstellung mit \x2F. Die Überprüfung mit strchr() erkennt dies aber.

In Zeile 2138 erfolgt der Wechsel von Root auf User mittels Aufruf der Funktion globus_gatekeeper_util_trans_to_user(). Diese benutzt, abhängig vom verwendetem System, Funktionen zum wechseln auf den User. Im Fall von Linux die Funktion setreuid(). Die Rückgabewerte werden ausreichend geprüft. Eine mögliche Code Injection Schwachstelle im weiteren Verlauf, wie sie zum Beispiel bei der verwendeten Funktion exec() existieren könnte, oder sonstige Schwächen, wäre damit nicht mehr kritisch, da keine Rechteausweitung mehr möglich ist.

Während der Untersuchung des Gatekeepers konnten auch sonst keine Schwachstellen entdeckt werden, die es einem Angreifer ermöglichen mit Hilfe der übermittelten Daten beliebigen Code auszuführen und damit eine Rechteausweitung zu erreichen oder sich Zugang zu vertraulichen Daten, wie fremden Proxyzertifikate und der dazugehörigen Schlüssel, zu verschaffen. Auch wurden keine Möglichkeiten gefunden die Erreichbarkeit des Gatekeepers, und damit des Gram Services, zu unterbinden oder die Authentifizierungs- und Autorisierungsprozesse zu umgehen.

Auswertung Accountfiles

Informationen für Abrechnungszwecke werden über die Funktion globus_gram_job_manager_request_acct() in das jeweilige Account- oder Logfile, sowie in die Syslogdatei, geschrieben. Sie befindet sich in der Datei globus_gram_job_manager.c die unter gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source zu finden ist.

```
01458 globus_gram_job_manager_request_acct(
01459     globus_gram_jobmanager_request_t *  request,
01460     const char *                        format,
01461     ... )
01462 {
01463     static const char *jm_syslog_id  = "gridinfo";
01464     static int         jm_syslog_fac = LOG_DAEMON;
01465     static int         jm_syslog_lvl = LOG_NOTICE;
01466     static int         jm_syslog_init;
01467     struct tm *curr_tm;
01468     time_t curr_time;
01469     va_list ap;
01470     int rc = -1;
01471     int fd;
01472     const char * gk_acct_fd_var = "GATEKEEPER_ACCT_FD";
01473     const char * gk_acct_fd;
01474     int n;
01475     int t;
01476     char buf[1024 * 128];
01477
01478     time( &curr_time );
01479     curr_tm = localtime( &curr_time );
01480
01481     n = t = sprintf( buf, "JMA %04d/%02d/%02d %02d:%02d:%02d ",
```

```
01482                curr_tm->tm_year + 1900,
01483                curr_tm->tm_mon + 1, curr_tm->tm_mday,
01484                curr_tm->tm_hour, curr_tm->tm_min,
01485                curr_tm->tm_sec );
01486
01487    va_start( ap, format );
01488
01489    /*
01490     * FIXME: we should use vsnprintf() here...
01491     */
01492
01493    n += vsprintf( buf + t, format, ap );
01494
01495    if (!jm_syslog_init)
01496    {
01497        const char *s;
01498
01499        if ((s = globus_libc_getenv( "JOBMANAGER_SYSLOG_ID"  )) != 0)
01500        {
01501            jm_syslog_id = *s ? s : 0;
01502        }
01503
01504        if ((s = globus_libc_getenv( "JOBMANAGER_SYSLOG_FAC" )) != 0)
01505        {
01506            if (sscanf( s, "%u", &jm_syslog_fac ) != 1)
01507            {
01508                jm_syslog_id = 0;
01509            }
01510        }
01511
01512        if ((s = globus_libc_getenv( "JOBMANAGER_SYSLOG_LVL" )) != 0) {
01513            if (sscanf( s, "%u", &jm_syslog_lvl ) != 1) {
01514                jm_syslog_id = 0;
01515            }
01516        }
01517
01518        if (jm_syslog_id)
01519        {
01520            openlog( jm_syslog_id, LOG_PID, jm_syslog_fac );
01521        }
01522
01523        jm_syslog_init = 1;
01524    }
01525
01526    if (jm_syslog_id)
01527    {
01528        char *p, *q = buf;
01529
01530        while ((p = q) < buf + n) {
01531            char c;
01532
01533            while ((c = *q) != 0 && c != '\n') {
01534                q++;
```

```
01535             }
01536
01537             *q = 0;
01538
01539             syslog( jm_syslog_lvl, "%s", p );
01540
01541             *q++ = c;
01542         }
01543     }
01544
01545     if (!(gk_acct_fd = globus_libc_getenv( gk_acct_fd_var )))
01546     {
01547         return -1;
01548     }
01549
01550     if (sscanf( gk_acct_fd, "%d", &fd ) != 1)
01551     {
01552         return -1;
01553     }
01554
01555     fcntl( fd, F_SETFD, FD_CLOEXEC );
01556
01557     if ((rc = write( fd, buf, n )) != n)
01558     {
01559         rc = -1;
01560     }
01561
01562     return rc;
01563 }
01564 /* globus_gram_job_manager_request_acct() */
```

Aufgerufen wird sie in den Funktionen globus_l_gram_job_manager_script_read(), die beim eintreffen von Antworten des Job Manager Skripts auf die Übermittlung von Anfragen gestartet wird, sowie in der Funktion globus_l_gram_job_manager_default_done(). Auch in der zweiten Funktion werden Rückgabewerte des Job Manager Skripts verarbeitet. Zu finden sind sie in der Datei globus_gram_job_manager_script.c unter gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source.

Ausschnitt aus globus_l_gram_job_manager_script_read():

```
00448         if(strcmp(script_variable, "GRAM_SCRIPT_JOB_ID") == 0)
00449         {
00450             const char * gk_jm_id_var = "GATEKEEPER_JM_ID";
00451             const char * gk_jm_id  = globus_libc_getenv(gk_jm_id_var);
00452             const char * gk_peer   = globus_libc_getenv("GATEKEEPER_PEER");
00453             const char * globus_id = globus_libc_getenv("GLOBUS_ID");
00454             uid_t uid = getuid();
00455             gid_t gid = getgid();
00456             const char *user = request->config->logname;
00457
00458             globus_gram_job_manager_request_acct(
```

```
00459                request, "%s %s for %s on %s\n", gk_jm_id_var,
00460                gk_jm_id  ? gk_jm_id  : "none",
00461                globus_id ? globus_id : "unknown",
00462                gk_peer   ? gk_peer   : "unknown");
00463
00464            globus_gram_job_manager_request_acct(
00465                request, "%s %s mapped to %s (%u, %u)\n", gk_jm_id_var,
00466                gk_jm_id  ? gk_jm_id  : "none",
00467                user, uid, gid);
00468
00469            globus_gram_job_manager_request_acct(
00470                request, "%s %s has %s %s manager type %s\n", gk_jm_id_var,
00471                gk_jm_id  ? gk_jm_id  : "none",
00472                script_variable, script_value,
00473                request->config->jobmanager_type);
00474        }
```

Ausschnitt aus globus_l_gram_job_manager_default_done():

```
01231    else if(strcmp(variable, "GRAM_SCRIPT_JOB_ACCT_INFO") == 0)
01232    {
01233        if(value != NULL && strlen(value) > 0)
01234        {
01235            const char *gk_jm_id_var = "GATEKEEPER_JM_ID";
01236            const char *gk_jm_id = globus_libc_getenv(gk_jm_id_var);
01237            const char *v = value;
01238            char *buf = malloc(strlen(value) + 1);
01239            char *b = buf;
01240            char c;
01241
01242            while ((*b++ = ((c = *v++) != '\\') ? c :
01243                           ((c = *v++) != 'n' ) ? c : '\n'))
01244            {
01245            }
01246
01247            globus_gram_job_manager_request_acct(
01248                request, "%s %s summary:\n%s\nJMA -- end of summary\n", gk_jm_id_var,
01249                gk_jm_id ? gk_jm_id : "none", buf);
01250
01251            free(buf);
01252        }
01253    }
```

Wie in Schritt eins erörtert, liefert das Job Manager Skript als Rückgabewert einen Job Identifier, den angefragten Status oder einen Fehlercode. Diese Informationen können von einem Client nicht derart beeinflusst werden, insbesondere nicht bezüglich ihrer Größe, dass sie Schaden anrichten könnten. Damit ist es auch nicht möglich die Variable buf aus Zeile 01476 zu überfüllen, obwohl, wie schon in dem Kommentar im Quelltext bemängelt, in Zeile 01493 nicht die Funktion vsnprintf sondern vsprintf, welche die Größe der Daten nicht begrenzt, verwendet wird.

Auswertung Job Manager

Um zu überprüfen, ob auf laufende Jobs von anderen Nutzern zugegriffen werden kann, wurde überprüft, wo die Authentifizierung erfolgt und ob der entsprechende Prozess umgangen werden kann. Interessant für einen Angreifer wäre es auch, fremde Jobs terminieren zu können, damit mehr Ressourcen für die eigenen zu Verfügung stehen. Die Authentifizierung erfolgt mittels der Funktion globus_gram_job_manager_authz_query(), die beim bearbeiten der Anfrage durch die Callbackfunktion globus_gram_job_manager_query_callback() aufgerufen wird. Der zugehörige Quelltext liegt in der Datei globus_gram_job_manager_query.c unter gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source/globus_gram_job_manager_query.c. Möglichkeiten den Prozess zu umgehen oder Einfluss auf laufende Jobs von anderen zu nehmen wurde nicht gefunden.

```
00123 globus_gram_job_manager_query_callback(
00124     void *                              arg,
00125     globus_gram_protocol_handle_t       handle,
00126     globus_byte_t *                     buf,
00127     globus_size_t                       nbytes,
00128     int                                 errorcode,
00129     char *                              uri)
00130 {
...
00225     rc = globus_gram_job_manager_authz_query(
00226             manager,
00227             handle,
00228             contact,
00229             query);
00230     if (rc != GLOBUS_SUCCESS)
00231     {
00232         goto authz_failed;
00233     }
```

Auswertung der Ereignisprotokollierung

Bei der Auswertung der Loggingprozesse wurde nach Schwachstellen gesucht, die es einem Angreifer ermöglichen, gezielt falsche Einträge in den Logdateien zu erzeugen bzw. existierende Einträge zu löschen. Dadurch könnten Aktionen von Angreifern verschleiert werden. Im Bereich des Gatekeepers wird mit dem Service Namen, der mit dem Gram Protokoll bei der Anfrage übermittelt wird, ein vom Client vorgegebener String geloggt.
gt5.0.2-all-source-installer/source-trees/gatekeeper/source/globus_gatekeeper.c:

```
01746     if (length > 256)
01747     {
01748         failure(FAILED_SERVICELOOKUP, "Service name malformed");
01749     }
01750
01751     notice3(LOG_NOTICE,
```

```
01752            "Requested service: %s %s",
01753            service_name,
01754            (got_ping_request) ? "[PING ONLY]" : "");
01755
01756    /* Don't allow the client to look for service files outside of the
01757     * service directory
01758     */
01759    if (strchr(service_name, '/') != NULL)
01760    {
01761        failure2(FAILED_SERVICELOOKUP, "Invalid service name %s", service_name);
01762    }
01763
01764    if ((rc = globus_gatekeeper_util_globusxmap(
01765                genfilename(gatekeeperhome,grid_services,service_name),
01766                NULL,
01767                &service_line)) != 0)
01768      {
01769            failure3(FAILED_SERVICELOOKUP,
01770                     "Failed to find requested service: %s: %d",
01771                     service_name, rc);
01772      }
```

Dies stellt aber kein wirkliches Problem dar, da die Variable durch Verwendung der Funktion sscanf() in Zeile 1578 keine Leerzeichen enthalten kann und dadurch keine komplexen Einträge erzeugt werden können.

gt5.0.2-all-source-installer/source-trees/gatekeeper/source/globus_gatekeeper.c:

```
01572    {
01573        char    save = http_message[length];
01574        char *  tmpbuf = (char *) malloc(length);
01575        char *  p;
01576
01577        http_message[length] = '\0';
01578        if ((1 != sscanf(http_message, "POST %s", tmpbuf)) ||
01579            (! (p = strchr(tmpbuf, '/'))))
01580        {
01581            failure(FAILED_SERVICELOOKUP,
01582                    "Unable to extract service name from incoming message\n");
01583        }
01584        http_message[length] = save;
01585
01586        if (strncmp(tmpbuf,"ping/",5)==0)
01587            got_ping_request = 1;
01588
01589        if ((mapping = strchr(tmpbuf, '@')) != NULL)
01590        {
01591            *mapping = '\0';
01592            mapping = strdup(++mapping);
01593        }
```

```
01594
01595         service_name = strdup(++p);
01596         free(tmpbuf);
01597     }
```

Im Job Manager wurde eine verdächtige Stelle in der Funktion globus_gram_job_manager_query_callback(), zu finden unter gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source/, entdeckt. Die Funktion wird aufgerufen, wenn Anfragen im Bezug auf existierende Jobs eintreffen.

```
00123 globus_gram_job_manager_query_callback(
00124     void *                              arg,
00125     globus_gram_protocol_handle_t       handle,
00126     globus_byte_t *                     buf,
00127     globus_size_t                       nbytes,
00128     int                                 errorcode,
00129     char *                              uri)
00130 {
00131     globus_gram_job_manager_t *         manager = arg;
00132     globus_gram_jobmanager_request_t *  request = NULL;
00133     char *                              query           = GLOBUS_NULL;
00134     char *                              rest;
00135     int                                 rc = 0;
00136     globus_gram_protocol_job_state_t    status = 0;
00137     int                                 exit_code = 0;
00138     int                                 job_failure_code = 0;
00139     globus_bool_t                       reply           = GLOBUS_TRUE;
00140     const char *                        contact;
00141
00142     globus_gram_job_manager_log(
00143         manager,
00144         GLOBUS_GRAM_JOB_MANAGER_LOG_INFO,
00145         "event=gram.query.start "
00146         "level=INFO "
00147         "uri=\"%s\" "
00148         "\n",
00149         uri);
00150
00151     if (manager->config->log_levels & GLOBUS_GRAM_JOB_MANAGER_LOG_TRACE)
00152     {
00153         char *                          querystring;
00154
00155         querystring = globus_gram_prepare_log_string((char *) buf);
00156
00157         globus_gram_job_manager_log(
00158             manager,
00159             GLOBUS_GRAM_JOB_MANAGER_LOG_TRACE,
00160             "event=gram.query.info "
00161             "level=TRACE "
00162             "uri=\"%s\" "
00163             "message=\"%s\" "
00164             "\n",
```

```
00165             uri,
00166             querystring ? querystring : "");
```

Mit der Variabel querystring kann eine Clienteingabe, die über die Variable buf an die Funktion übergeben wird, in die Logdateien geschrieben werden. Allerdings wird hier mit Hilfe der Funktion globus_gram_prepare_log_string() die Clienteingabe derart modifiziert, dass eine Manipulation der Logdatei sofort erkannt werden kann, da eventuell vorkommende Steuerzeichen verändert werden.

gt5.0.2-all-source-installer/source-trees/gram/jobmanager/source/logging.c:

```
00140 char *
00141 globus_gram_prepare_log_string(
00142     const char *                        instr)
00143 {
00144     char *                              outstr;
00145     int                                 i = 0;
00146     if (instr == NULL)
00147     {
00148         return NULL;
00149     }
00150     outstr = malloc(2*strlen(instr) + 1);
00151     if (outstr == NULL)
00152     {
00153         return NULL;
00154     }
00155
00156     while (*instr != 0)
00157     {
00158         if (*instr == '\n')
00159         {
00160             outstr[i++] = '\\';
00161             outstr[i++] = 'n';
00162             instr++;
00163         }
00164         else if (*instr == '\r')
00165         {
00166             outstr[i++] = '\\';
00167             outstr[i++] = 'r';
00168             instr++;
00169         }
00170         else if (*instr == '\\')
00171         {
00172             outstr[i++] = '\\';
00173             outstr[i++] = '\\';
00174             instr++;
00175         }
00176         else if (*instr == '"')
00177         {
00178             outstr[i++] = '\\';
00179             outstr[i++] = '"';
```

```
00180             instr++;
00181         }
00182         else
00183         {
00184             outstr[i++] = *(instr++);
00185         }
00186     }
00187     outstr[i++] = '\0';
00188
00189     return outstr;
```

Im weiteren Verlauf der Untersuchung würden keine Möglichkeiten gefunden die Logdateien gezielt zu manipulieren oder gar zu löschen.

Auswertung des Scheduler Event Generators

Der Scheduler Event Generator läuft mit privilegierten Rechten und stellt deshalb für einen Angreifer ein auf den ersten Blick attraktives Ziel dar. Die Untersuchung erfolgt erst jetzt, da der Scheduler Event Generator nicht standardmäßig aktiviert ist.

Er erzeugt Events nur auf Basis der Einträge in den Logdateien der jeweiligen LRM und diese liegen nicht unter der Kontrolle von GRAM. Ein Angreifer kann also über GRAM selbst Einträge nicht gezielt verändern. Sollte er aber, z.B. über die abgesetzten Jobs, die Logeinträge manipulieren können, so wäre er in der Lage die laufenden Jobs zu beeinflussen. Er könnte beispielsweise den Status auf DONE setzen. Das SEG LRM Module würde den Eintrag lesen, über den SEG den Status in der Anfrage ändern und die Statusmaschine anregen. Ein Eintrag in das Accountingfile und eine Benachrichtigung an die Registrierten Clients wären die Folge. Außerdem werden daraufhin die Statusmeldungen über den SEG deaktiviert und korrekte Statusänderungen würden den Job Manager nicht mehr erreichen. Der Job selbst würde aber weiter laufen. Setzt der Angreifer den Status auf FAIL, so könnte er die Clients dazu anregen, den Auftrag wiederholt abzusenden. Damit könnte er indirekt die Last erhöhen.

Exemplarisch wurde das Load Sharing Facility (lsf) Modul genauer untersucht. Dabei wurden aber keine Möglichkeiten gefunden, über gefälschte Einträge eigenen Code ausführen zu lassen bzw. Daten zu schreiben oder zu lesen.

Beim Einsatz des SEG hängt die Sicherheit also wesentlich vom eingesetzen Local Resource Manager ab.

4.5 Diskussion der Ergebnisse

Die Ergebnisse und Erfahrungen aus der vorangegangenen Untersuchung werden nun noch einmal zusammengefasst und bilden die Basis, um die Verwundbarkeit des Globus Resource and Allocation Management 5 und die First Priciples Vulnerability Assessment Methode selbst zu bewerten. Kriterien für die Verwundbarkeit von GRAM5 sind Anzahl und Schwere von gefundenen Schwachstellen. Für die Methode selbst ist die Fähigkeit den Aufwand einer Untersuchung zu reduzieren und die Erkennungsrate maßgeblich.

Die Untersuchung von GRAM5 mit der First Principles Vulnerability Assessment Methode hat keine kritischen Schwachstellen aufgezeigt. Dies liegt aber weniger an der Methode, als vielmehr an GRAM5. Es verteilt Aufgaben klar strukturiert auf unterschiedliche Komponenten und nutzt, um kritische Aufgaben auszulagern, vom Globus Toolkit angebotene Komponenten über APIs.

Durch die im Gatekeeper erfolgte Authentifizierung und Autorisierung sowie dem dabei erzeugten Sicherheitskontext, der in der gesamten weiteren Clientkommunikation eingesetzt wird, ist die Vertraulichkeit gewährleistet. Die kritischen Arbeitsschritte hierbei werden an die Grid Security Infrastructure abgegeben. Es zeigten sich keine Schwächen, die es ermöglichten diese Prozesse zu umgehen und somit unberechtigt Ressourcen zu Nutzen oder auf fremde Ressourcen und Dateien zuzugreifen. Auch bei Anfragen zu laufenden Prozessen, die direkt an den vom jeweiligen Job Manager erzeugten Listener gestellt werden, konnte keine Möglichkeit gefunden werden, um unbefugt an Daten zu gelangen oder laufende Jobs sonst wie zu manipulieren.

gt5.0.2-all-source-installer/source-trees/gatekeeper/source/globus_gatekeeper.c:

```
01487     major_status = globus_gss_assist_accept_sec_context(
01488         &minor_status,
01489         &context_handle,
01490         credential_handle,
01491         &client_name,
01492         &ret_flags,
01493         NULL,            /* don't need user_to_user */
01494         &token_status,
01495         &delegated_cred_handle,
01496         globus_gss_assist_token_get_fd,
01497         (void *)stdin,
01498         globus_gss_assist_token_send_fd,
01499         (void *)fdout);

...

01629     result = globus_gss_assist_map_and_authorize(context_handle,
01630                                                  service_name,
01631                                                  mapping,
01632                                                  identity_buffer, 256);
```

Bereiche die mit privilegierten Rechten laufen sind auf ein Minimum eingeschränkt und damit werden die Angriffsflächen für eine vertikale oder horizontale Rechteausweitung redu-

ziert. Im Gatekeeper werden beim Start einer Kommunikation alsbald die Authentifizierungs- und Autorisierungsprozesse durchgeführt sowie auf die Nutzerrechte der jeweiligen User gewechselt.

Um zu verhindern, dass die Verfügbarkeit des GRAM Service gänzlich gestört wird, ist auch hier der kritische Bereiche auf einen Teil des Gatekeepers eingegrenzt. Soweit es die Untersuchung zeigte, wurden hier alle Clienteingaben ausreichend validiert sowie Speichergrößen richtig berechnet. Damit sollten Pufferüberläufe in diesem Bereich nicht möglich sein. Läuft der Gatekeeper als Dämon, ist die Anzahl gleichzeitiger Anfragen begrenzt, wodurch eine mögliche Überlast verhindert wird. Wenn er über einen Superserver gestartet wird, muss dieser dafür Sorge tragen.

Um Aktionen klar nachvollziehbar zu machen, werden Ereignisse intensiv in Log- und Abrechnungsdateien protokolliert. Dadurch können auch mögliche Angriffsversuche erkannt werden. Es zeigten sich bei der Untersuchung keine Möglichkeiten für Clients gefälschte Einträge zu erzeugen. Wo es möglich ist, werden Clienteingaben nicht direkt geloggt und an den wenigen gefundenen Stellen an denen doch Eingaben Einzug in die Files finden, wird vorher sichergestellt, dass keine gültigen Einträge gefälscht werden können.

Der Einsatz des Scheduler Event Generators, der einen gewissen Performanzgewinn bringt, stellt ein Risiko dar. Die Sicherheit ist hier vom eingesetzten Local Resource Manager abhängig.

Negativ aufgefallen ist, dass keine Behandlungsroutine für gesperrte Zertifikate existiert. Falls diese als gestohlen gemeldet werden, so ist es zwar nicht mehr möglich neue Aufträge abzusetzen, aber bereits in Bearbeitung befindliche laufen weiter.

Ansonsten sind nur stilistische Mängel zu erkennen, diese gefährden die Sicherheit von GRAM5 aber in keinster Weise. So sind einzelne Codesegmente nur mit if(0) Konstrukten auskommentiert, statt sie vor der Veröffentlichung der Quelltexte zu entfernen. Auch sind Kommentare aus Codereviews in den Quelltexten verblieben. Diese legen, zusammen mit der Tatsache, dass keine kritischen Schwachstellen gefunden wurden, die Vermutung nahe, dass die Komponente schon intensiv auf mögliche Schwächen untersucht wurde.

gt5.0.2-all-source-installer/source-trees/gatekeeper/source/globus_gatekeeper.c:

```
01883 #if 0
01884         char *     proxyfile;
01885         int        fd, i;
01886         size_t     len, bufsize;
01887
01888         /*
01889          * DEE this is no longer needed as the delegated cred is
01890          * returned, and proxy is still in memory
01891          * failure will cleanup the delegated cred.
01892          */
01893         if ( ((proxyfile = getenv("X509_USER_DELEG_PROXY")) != NULL)
```

```
01894              && ((fd = open(proxyfile, O_RDWR, 0600) >= 0))  )
01895         {
01896             len = lseek(fd, 0, SEEK_END);
01897             lseek(fd, 0, SEEK_SET);
01898             bufsize = sizeof(tmpbuf);
01899             for (i=0; i<bufsize; i++)
01900                 tmpbuf[i] = 0;
01901
01902             for (i=0; i<len; )
01903                 i += write(fd, tmpbuf, MIN(bufsize, len-i));
01904
01905             close(fd);
01906             unlink(proxyfile);
01907         }
01908 #endif
```

gt5.0.2-all-source-installer/source-trees/gatekeeper/source/globus_gram_job_manager_request.c:

```
01490      * FIXME: we should use vsnprintf() here...
01491      */
01492
01493     n += vsprintf( buf + t, format, ap );
```

Auf Basis der vorangegangenen Untersuchung der Komponente mit Hilfe der First Principles Vulnerability Assessment Methode, kann diese also in der Standardkonfiguration als sicher angesehen werden. Dies war von einer Komponente des Globus Toolkits, welches als de facto Standard im Grid Bereich angesehen wird [46], nicht anders zu erwarten.

Ein großes Problem bei der Suche nach Schwachstellen ist oft die Größe und Komplexität der untersuchten Systeme. Die Mittel die für eine Untersuchung aufgebracht werden können sind meist limitiert. Es können nicht unbegrenzt Mitarbeiter dafür abgestellt werden und oft muss eine Untersuchung in einem sehr kurzen Zeitrahmen erfolgen, da die Software schnell auf den Markt gebracht werden soll. Entweder da diese dringend gebraucht wird oder weil finanzielle Investitionen Gewinne abwerfen sollen. Dabei dürfen aber kritische Schwächen nicht übersehen werden. Mit diesem Tradeoff muss eine Methode zur Suche nach Schwachstellen umgehen können.

Die First Principles Vulnerability Assessment Methode eignet sich, wie andere Methoden auch, nur dazu Schwachstellen zu erkennen aber nicht die Abwesenheit selbiger zu beweisen. Sie schafft es jedoch, den erforderlichen Aufwand bei einer Untersuchung zu reduzieren und gezielt auf kritische Bereiche zu richten. Bei begrenzten Mittel können beispielsweise große Teile des Job Managers von einer Untersuchung ausgenommen werden, da durch die Ergebnisse der Schritte eins bis drei ersichtlich wird, dass ein erfolgreicher Angriff nur einen geringen Schaden mit sich bringen würde. Ein weiteres Beispiel ist das Gridmapfile, welches bei einer ersten Betrachtung ein klassisches Zwischenziel für weitere Angriffe darstellt. Hier zeigt sich in den Analyseschritten, dass nur über die Grid Security Infrastructure zugriffen

wird und keine Möglichkeit besteht, das File über die GRAM5 Komponenten zu verändern.

Ein kurzer Vergleich mit den Ergebnissen die das automatische Tool zum Finden von Schwachstellen „flawfinder“ bei der Untersuchung des Gatekeepers liefert zeigt, dass mit der First Principles Vulnerability Assessment Methode keine Bereiche übersehen wurden. Vielmehr wurden auch kritische Funktionen wie fd_set untersucht, die von flawfinder nicht erkannt wurden.

Die Architekturanalyse, die im Gegensatz zu der auch vorgestellten Threat Modeling Methode, welche nur ausgewählte Teile genauer betrachtet, die ganze GRAM5 Komponente untersucht, hilft gerade unerfahrenen Analysten ein gutes Verständnis für die Arbeitsweise der untersuchten Anwendung zu erlangen. Dadurch können logische Fehler im Design leichter erkannt werden.

Während der Untersuchung von GRAM5 hat sich aber auch offenbart, dass die Methode keineswegs nur rein linear durchgeführt werden kann. Schon in Schritt eins zeigt sich, dass die Erzeugung von Kindprozessen im Job Manager von der Identität und damit indirekt von den Rechten abhängig ist. Die Betrachtung der Rechte wird damit vorweg genommen. Sollte sich in Schritt vier, also in der Auswertung der Komponenten, außerdem herausstellen, dass eine horizontale Rechteausweitung möglich ist, könnten damit unkritische Bereiche zu kritischen werden. Die Auswahl der Ziele müsste neu überdacht werden, da sich Punkt drei, Vertrauen und Rechte, dadurch verändern würde. Die Methode ist daher in Teilen als iterativer Prozess zu sehen.

Die Schritte eins bis drei der Methode sollten im wesentlichen auf Basis der Quelltexte erfolgen, da Dokumentationen oft unvollständig, veraltet oder einfach falsch sein können und Informationen von Entwicklern von ihrer Subjektiven Meinung beeinflusst sind. Bei der Ausführung der Schritte auf Grundlage des Quellcodes wird man aber leicht verleitet Annahmen vorweg zu nehmen und die Reihenfolge der Arbeitsschritte unnötig aufzuweichen, wodurch die Qualität der Untersuchung leiden kann. Deshalb ist darauf zu achten, Informationen die einem bei der ersten Untersuchung des Codes unnötig erscheinen dennoch im Analyseprozess aufzunehmen, da diese später wichtig sein können.

Obwohl bei der Untersuchung keine kritischen Schwachstellen gefunden wurden, zeigen die dabei gemachten Erfahrungen, dass die First Principles Vulnerability Assessment Methode die geforderten Anforderungen erfüllt. Sie kann den Aufwand einer Untersuchung erheblich reduzieren und ist offensichtlich in der Lage kritische Schwachstellen zu entdecken. Auch ist sie einfach gegliedert und kann in der Praxis leicht umgesetzt werden.

5 Zusammenfassung und weiterführende Arbeiten

Dieses Kapitel fasst die Ergebnisse und Erkenntnisse dieser Arbeit kurz zusammen. Auch enthält es Anregungen für weitere Arbeiten in diesem Themengebiet.

Da Grid Infrastrukturen einen wichtigen Bereich in der Forschung darstellen und zunehmend mehr Einzug in die Wirtschaft halten, rücken die Begriffe Zuverlässligkeit und Sicherheit immer mehr in den Mittelpunkt. Eine Anwendung ist verwundbar, wenn sie Schwachstellen enthält die von möglichen Angreifern ausgenutzt werden können um an wertvolle Ressourcen zu gelangen. Dies sind beispielsweise sensible Daten oder erlangte Rechte, kann aber auch die Schädigung des Ansehens von Anderen sein. Da Grid Umgebungen aufgrund ihrer Dynamik, also dem häufigen Wechsel von Mitgliedern, Gruppen und Ressourcen, besonders gefährdet sind, wird eine Methode benötigt, welche Schwachstellen in einer Grid Middleware zuverlässig entdecken kann. Zwar gibt es mit der Threat Modeling schon eine in der Praxis eingesetzte Methode, diese arbeitet aber auf Basis bereits bekannter Schwachstellen und Bedrohungen und ist damit nicht in der Lage neue Schwächen zu entdecken. Es existieren auch etliche Programm für das automatisierte Aufspüren von Schwachstellen in Quelltexten oder in lauffähigen Anwendungen, aber auch diese weisen das selbe Problem auf. Auch können mit Programmen, die lauffähige Anwendungen untersuchen, die erkannten Schwächen kaum im Quellcode lokalisiert werden. Dies ist aber nötig, falls die Ergebnisse nicht nur dazu genutzt werden sollen bestehende Anwendungen bezüglich ihrer Verwundbarkeit miteinander zu vergleichen, sondern diese auch zu schließen.

Die Universitäten von Wisconsin und Barcelona haben mit der First Principles Vulnerability Assessment Methode einen Prozess vorgestellt der die bestehende Lücke schließen könnte. Mit der Methode wurden von den Entwicklern zwar schon einige Anwendungen im Grid Umfeld untersucht, die veröffentlichen Ergebnisse lassen sich aber nicht nutzen um die Methode unabhängig zu bewerten. Kriterien dafür sind nicht nur eine hohe Erkennungsrate sondern auch die Fähigkeit den zu betreibenden Aufwand zu minimieren. Dies ist nötig, da die Mittel, die für Untersuchungen bereitgestellt werden können, meist begrenzt sind und eine zeilenweise Untersuchung von komplexen Anwendungen daher nicht durchführbar ist.

Die First Principles Vulnerability Assessment Methode leitet die Untersuchung in fünf Schritten, wobei die ersten drei dazu dienen ein tieferes Verständnis für das untersuchte System zu entwickeln und wertvolle Ziele zu identifizieren. Dazu werden die Hauptkomponenten und deren Interaktionen sowie die von diesen benutzten Ressourcen bestimmt. Zusätzlich werden die Rechte mit denen sie ausgestattet sind identifiziert und das Vertrauen das einzelnen Ressourcen zugestanden werden kann diskutiert. Im vierten Schritt werden die Ziele identifiziert und priorisiert, um ausgehend von ihnen den Quelltext manuell nach Schwachstellen zu untersuchen. Abschließend erfolgt die Veröffentlichung der Ergebnisse.

Um die First Principles Vulnerability Assessment Methode zu bewerten, wurde ein Teil des Globus Toolkit 5, einer gängigen Grid Middleware, untersucht. Dieses stellt Komponenten bereit, die den Aufbau und Betrieb von Gridumgebungen ermöglichen sowie die Entwicklung von Anwendungen im Gridumfeld unterstützen. Die Wahl viel auf das Grid Resource Allocation Management 5, da dieses grundlegende Veränderungen gegenüber seinem Vorgänger GRAM4 erfahren hat. Es wendet sich von Web Services ab und setzt nun, wie schon GRAM2, auf statusbehaftete Ressourcen.

Bei der durchgeführten Untersuchung der Quelltexte von GRAM5 wurden keine schwerwiegende Schwächen entdeckt. Aufgaben werden, so weit wie möglich, in nicht privilegierten Komponenten ausgeführt und kritische Funktionalitäten, wie der Authentifizierung- und Autorisierungsprozess, werden ausgelagert, indem sie über APIs an Komponenten wie die Grid Security Infrastruktur übergeben werden. In den untersuchten Codesegmenten wurden keine Clienteingaben gefunden, die ungeprüft oder unverändert als Argumente verwendet werden. Einzig der optionale Einsatz des Scheduler Event Generators birgt ein gewisses Risiko. Im ausgeliefertem Quellcode verbliebene Reviewkommentare deuten darauf hin, dass die Anwendung bereits intensiv auf Schwachstellen hin untersucht worden ist. Auf Basis der hier durchgeführten Untersuchung mit Hilfe der First Principles Vulnerability Assessment Methode scheint es, dass GRAM5 in der Standardkonfiguration als sicher angesehen werden kann. In zukünftigen Arbeiten könnte eine Methode gesucht werden, um den Umgang von GRAM5 mit gesperrten Zertifikaten zu regeln.

Um die hier vorgestellte Methode besser bezüglich ihrer Erkennungsrate mit anderen vergleichen zu können, wäre es sinnvoll eine noch nicht auf Schwachstellen hin untersuchte Anwendung parallel mit ihr und einer anderen Methode, wie etwa dem Threat Modeling, zu analysieren und die Ergebnisse miteinander zu vergleichen.

Dennoch konnten bei der Durchführung Erkenntnisse gewonnen werden die zeigen, dass mit der First Principles Vulnerability Assessment Methode ein Prozess formuliert wurde, der sich gut zur Suche nach Schwachstellen in komplexen, verteilten Systeme eignet und welcher auch mit der Dynamik von Gridumgebungen umgehen kann.

Sie adaptiert im wesentlichen das Vorgehen von Angreifern, welche nach Schwächen im Design oder nach sonstigen Fehlern suchen, die es ermöglichen das System für eigene Zwecke zu missbrauchen bzw. anderen Schaden zuzufügen. Mit ihr entdeckte Schwächen werden daher auch mit hoher Wahrscheinlichkeit von Angreifern ausfindig gemacht. Im Gegensatz zu dem in dieser Arbeit auch angesprochenen Threat Modeling Ansatz erfolgt die Untersuchung ohne sich vorher auf mögliche Angriffsziele oder -methoden festzulegen. Dies ermöglicht es einen größeren Bereich zu betrachten und bisher unbekannte Schwachstellen zu finden. Ein kreativer Angreifer nimmt gerne einen „Umweg“ über ein im ersten Moment nicht lohnenswert erscheinendes Ziel in Kauf, um ein System zu kompromittieren. Diese Zwischenziele können aber durch die Methode entdeckt und untersucht werden.

Die First Principles Vulnerability Assessment Methode ermöglicht es durch die Auswahl und Priorisierung der Ziele den erforderlichen Aufwand einer Untersuchung zu reduzieren. Komponenten auf die Angreifer keinen direkten oder indirekten Einfluss nehmen können

werden erkannt und können von der Untersuchung ausgenommen werden. Die Auswertung von Zielen mit geringem Wert kann aufgeschoben werden, da erfolgreiche Angriffe auf diese nur einen geringen Schaden verursachen würden. Die ganze Aufmerksamkeit kann auf Ziele mit hohem Wert gerichtet werden und wenn bei einer Untersuchung Schwachstellen gefunden werden, sind sie dann meist von kritischer Natur.

Obwohl die Methode es ermöglicht den Aufwand zu verringern, bleibt eine manuelle Codeanalyse ein aufwendiger Prozess der viel Erfahrung und Kreativität vom Untersuchenden verlangt. Gerade unerfahrene Analysten werden daher oft auf unterstützende Tools zurückgreifen wollen, wie sie von Microsoft zur Durchführung der Threat Modeling Methode bereits bereitgestellt werden. Um die First Principles Vulnerability Assessment Methode zu etablieren, sollten daher in naher Zukunft Programme entwickelt werden, die die Ausführung einer Untersuchung unterstützen.

Zusätzlich sollten, wie von den Entwicklern vorgeschlagen, mit der Methode erzielte Untersuchungsergebnisse dazu verwendet werden, automatische Analyseprogramme zu verbessern. Dies könnte über identifizierte Angriffsvektoren erfolgen oder Programme wie flawfinder könnten um eine automatische Datenflussanalyse erweitert werden, was es ermöglichen würde die False-Positive Rate zu verringern.

Bei alledem darf nicht vergessen werden, dass die statische Codeanalyse nur einen Teilprozess bei der Entwicklung sicherer Anwendungen darstellt. Schon beim Design und bei der Implementierung muss auf die Sicherheit geachtet werden. Daraufhin sollte eine statische Codeanalyse folgen um Schwachstellen zu identifizieren und diese anschließend schließen zu können. Vor der endgültigen Auslieferung ist es sinnvoll, Penetrationstests, mit Verfahren wie dem auch von der Universität von Wisconsin entwickelten und bereits etablierten Fuzzing, durchzuführen.

Ist es ausschließlich das Ziel bestehende Anwendungen bezüglich ihrer Verwundbarkeit zu vergleichen, so ist eine statische Analyse wohl zu aufwendig, da eine Identifizierung der Schwächen im Code nicht benötigt wird, sondern nur das Vorhandensein selbiger in eine Wertung einfließen muss. Dynamisch Verfahren wie das Fuzzing können diese Aufgaben mit weniger Einsatz erfüllen.

Abbildungsverzeichnis

Literaturverzeichnis

[1] Foster, Kesselman C. Tuecke S. I.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In: *International Journal of Supercomputer Applications* 15 (2001)

[2] *Worldwide LHC Computing Grid*. Februar 2011. – http://lcg.web.cern.ch/

[3] James A. Kupsch, Eduardo C. Barton P. Miller M. Barton P. Miller ; Heymann, Elisa: First Principles Vulnerability Assessment (extended version). In: *MIST Project Technical Report* (2009)

[4] *Fuzz Testing of Application Reliability*. Februar 2011. – http://pages.cs.wisc.edu/~bart/fuzz/

[5] *Open Web Application Security Project, Category:Vulnerability*. Januar 2011. – http://www.owasp.org/index.php/Category:Vulnerability

[6] Shirey, R.: *RFC2828 - Internet Security Glossary*

[7] *Computer Emergency Response Team (CERT)*. Februar 2011. – http://www.cert.org/

[8] *Globus Allicance, Category:About the Globus Alliance*. Januar 2011. – http://www.globus.org/alliance/about.php

[9] *Globus Toolkit, Category:About the Globus Toolkit*. Januar 2011. – http://www.globus.org/toolkit/about.html

[10] *Globus Toolkit, Category:Globus Toolkit 5.0.2 Release Manuals*. Januar 2011. – http://www.globus.org/toolkit/docs/5.0/5.0.2/

[11] R. Housley, W. Ford D. S. W. Polk P. W. Polk: *RFC2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile*

[12] S. Tuecke, D. Engert L. Pearlman M. T. V. Welch W. V. Welch: *RFC 3820 - Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*

[13] Linn, J.: *RFC 2743 - Generic Security Service Application Program Interfac*

[14] Wray, J.: *RFC 2744 - Generic Security Service API Version 2 : C-bindings*

[15] S. Meder, U. Chicago S. Tuecke D. E. V. Welch W. V. Welch: *GSS-API Extensions*

[16] *MyProxy*. Februar 2011. – http://grid.ncsa.illinois.edu/myproxy/

[17] *SimpleCA*. Februar 2011. – http://www.vpnc.org/SimpleCA/

[18] *OpenSSL*. Februar 2011. – http://www.openssl.org/

[19] *OpenSSH*. Februar 2011. – http://www.openssh.com/

[20] *GSI-Enabled OpenSSH*. Februar 2011. – http://grid.ncsa.illinois.edu/ssh/

[21] W. Allcock, R. Kettimuthu M. Link C. Dumitrescu I. Raicu I. F. J. Bresnahan B. J. Bresnahan: The Globus Striped GridFTP Framework and Server. In: *Proceedings of Super Computing* (2005)

[22] Allcock, W.: *GFD.020 GridFTP extensions - GridFTP: Protocol Extensions to FTP for the Grid*

[23] J. Postel, J. R.: *RFC 959 - FILE TRANSFER PROTOCOL (FTP)*

[24] *Condor - High Throughput Computing*. Februar 2011. – http://www.cs.wisc.edu/condor/

[25] *OASIS Web Services Resource Framework (WSRF) TC*. Februar 2011. – http://www.oasis-open.org/committees/tc_home.php

[26] David Booth, Francis McCabe Eric Newcomer Michael Champion Chris Ferris David O. Hugo Haas H. Hugo Haas: *Web Services Architecture - W3C Working Group Note 11 February 2004*

[27] *GT 5.0.0 Release Notes*. Februar 2011. – http://www.globus.org/toolkit/docs/5.0/5.0.0/rn/

[28] *About The Open Web Application Security Project*. Februar 2011. – http://www.owasp.org/index.php/About_OWASP

[29] *National Vulnerability Database*. 2011. – http://nvd.nist.gov/

[30] *msdn, Threat Modeling Web Applications*. Februar 2011. – http://msdn.microsoft.com/en-us/library/ms978516.aspx

[31] *msdn, Chapter 3 - Threat Modeling*. Februar 2011. – http://msdn.microsoft.com/en-us/library/ff648644.aspx

[32] *msdn, The STRIDE Threat Model*. Februar 2011. – http://msdn.microsoft.com/en-us/library/ms954176.aspx

[33] *msdn, Microsoft SDL Threat Modeling Tool*. Februar 2011. – http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx

[34] *lint man page*. Februar 2011. – http://www.manpages.unixforum.co.uk/man-pages/unix/freebsd-6.2/1/lint-man-page.html

[35] *splint - Annotation-Assisted Lightweight Static Checking*. Februar 2011. – http://splint.org/

[36] *RATS*. Februar 2011. – https://www.fortify.com/ssa-elements/threat-intelligence/rats.html

[37] *Flawfinder*. 2011. – http://www.dwheeler.com/flawfinder/

[38] *Nessus*. Februar 2011. – http://www.nessus.org/

[39] *Wikipedia - List of tools for static code analysis*. Februar 2011. – http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

[40] Misha Zitser, Tim L. Richard Lippmann L. Richard Lippmann: *Testing static analysis tools using exploitable buffer overflows from open source code*. http://portal.acm.org/citation.cfm?id=1041685.1029911, nov 2004. – accessed: 13.02.2011

[41] R. Fielding, J. Gettys J. Mogul H. Frystyk L. Masinter P. Leach T. Berners-Lee UC. I. UC. Irvine: *RFC2616 - Hypertext Transfer Protocol – HTTP/1.1*

[42] *man exec(3)*. Februar 2011. – http://linux.die.net/man/3/exec

[43] *Advisories : multiple applications fd_set structure bitmap array index overflow*. Februar 2011. – http://securityvulns.com/advisories/sockets.asp

[44] *listen(2) - Linux man page*. 2011. – http://linux.die.net/man/2/listen

[45] *getpeername(3) - Linux man page*. Februar 2011. – http://linux.die.net/man/3/getpeername

[46] *Globus Toolkit 2.0 Wins R and D 100 Award*. 2011. – http://www.globus.org/toolkit/news/rd100.html