Distack: Framework zur einfachen Entwicklung von Mechanismen zur lokalen und verteilten Angriffserkennung

Christoph P. Mayer
Institut für Telematik
Universität Karlsruhe (TH)
Germany
mayer@tm.uka.de

Thomas Gamer
Institut für Telematik
Universität Karlsruhe (TH)
Germany
gamer@tm.uka.de

Ende 2007 wurde ein 18-jähriger Neuseeländer festgenommen, welcher ein Botnetz mit einer Größe von ca. 1,3 Millionen Zombies steuerte [1]. Das Botnetz verursachte durch die damit ausgeführten *Distributed Denial of Service*-Angriffe (DDoS-Angriffe) einen geschätzten Schaden von 13,5 Millionen Euro. Der Vorfall macht deutlich, dass DDoS-Angriffe auch heute noch ein wichtiges Thema im Bereich Netzwerksicherheit darstellen und effektive Gegenmaßnahmen noch immer fehlen. Gründe für das Fehlen effektiver Gegenmaßnahmen sind unter anderem das unvollständige Verständnis über das Verhalten von DDoS-Angriffen aus einer netzglobalen Sicht sowie die Schwierigkeit, neue Methoden zur lokalen und verteilten Angriffserkennung schnell und einfach zu entwickeln und zu evaluieren. Forscher, die neue Methoden zur Angriffserkennung entwickeln und evaluieren möchten, stehen vor initialen Problemen wie z. B. Zugriff auf die Netzwerkschnittstelle, Verwaltung von Tracedateien, Parsen von Protokollen und Kommunikation zwischen verteilten Instanzen der Angriffserkennung. Weiterhin ist es nicht ohne weiteres möglich Mechanismen, welche für reale Umgebungen entwickelt wurden, in simulierten Umgebungen transparent zu betreiben.

Das *Distack-Framework* wurde mit dem Ziel entwickelt Forschern die Möglichkeit zur einfachen Entwicklung und Evaluierung von Mechanismen zur lokalen und verteilten Angriffserkennung bereit zu stellen. Der Leitgedanke von *Distack* ist, dass sich Forscher auf die eigentlichen Mechanismen zur Angriffserkennung konzentrieren können statt aufwendige und fehleranfällige Verwaltungsaufgaben – wie beispielhaft oben genannt – implementieren zu müssen.

Die Architektur des *Distack-Framework* ist in Abbildung 1 gezeigt. Der *NetworkManager* abstrahiert vom unterliegenden Netzwerk sowie von der Laufzeitumgebung. Dadurch wird eine einheitliche Schnittstelle für den übergeordneten *ModuleManager* geschaffen. Somit ist die transparente Verwendung von Online- und Offline-Verkehr sowie von simulierten Netzwerken möglich. Die eigentliche Integration von Mechanismen zur Angriffserkennung oder Analyse von Netzverkehr findet im ModuleManager statt. Leichtgewichtige *Module* implementieren abgeschlossene Aufgaben und stellen somit funktionale Einheiten dar. Durch die strikte Trennung von Aufgaben ist eine hohe Wiederverwendbarkeit von Modulen gewährleistet. Zusammengehörige Module können sequentiell durch die XML-basierte Konfiguration in so genannte *Channels* kombiniert werden. Jeder vom NetworkManager in den ModuleManager eingelesene Netzwerkframe wird an jeden aktuell geladenen Channel weitergeleitet. Der Netzwerkframe wird dann entsprechend der sequentiellen Reihenfolge der Module in dem entsprechenden Channel abgearbeitet. Durch die im Framework vorhandenen objektorientierten Protokoll-Parser ist es sehr einfach den Inhalt eines Netzwerkframe zu analysieren. Weiterhin reduziert sich durch die global verwendeten Protokoll-Parser der Programmcode von Modulen. Außerdem wird durch diesen Ansatz die Robustheit der Analysen erhöht: Protokoll-Parser werden einmalig vom Framework implementiert, wodurch nicht jedes Modul eigene und fehlerträchtige Protokoll-Parser betreiben muss. Jedes Modul hat weiterhin die Möglichkeit die Verarbeitung eines Netzwerkframe in seinem Channel abzubrechen. Somit ist die einfache Realisierung von Sampling- und Filtering-Modulen möglich.

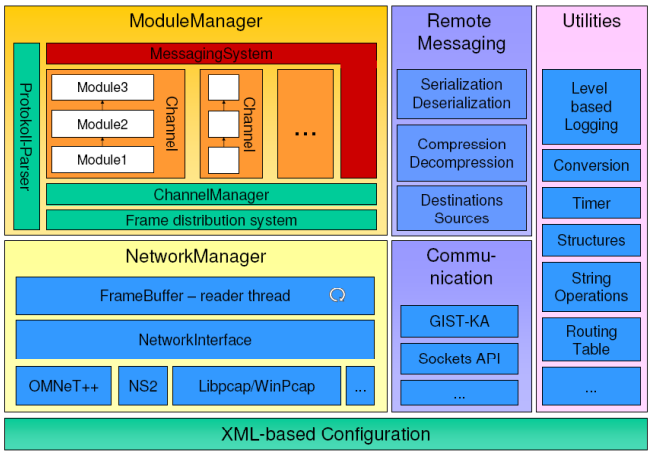

Abbildung 1: Architektur des *Distack Framework*

Neben der Gruppierung von Modulen in Channels ist die Gruppierung von Channels zu Stufen möglich. Hierdurch kann ein Granularitäts-basierter Ansatz (siehe auch [2]) zum Betrieb einer Angriffserkennung auf ressourcenschwachen Systemen realisiert werden: Im Normalzustand werden nur Module betrieben, welche eine Grundüberwachung des Verkehrs vornehmen. Falls ein Verdacht auf einen Angriff vorliegt, wird die nächsthöhere Stufe mit allen in ihr konfigurierten Channels geladen. Module sind in externen Bibliotheken realisiert. Durch die XML-basierte Konfiguration werden Modulen die für ihre Mechanismen notwendigen Parametern übergeben und das Modul mit einem Namen instanziiert. Der vergebene Name wird bei der Konfiguration der Channels verwendet um die Instanz des Moduls zu einem Channel hinzuzufügen. Somit ist die mehrfache Verwendung eines Moduls möglich. Dies macht z. B. bei Sampling- und Filtering-Modulen Sinn.

Da die Funktionalität von Modulen begrenzt ist um eine hohe Wiederverwendbarkeit entsprechend einem Baukastenprinzip zu erreichen, müssen zusätzlich interne Mechanismen zur Kommunikation zwischen Modulen bereitgestellt werden. Hierzu bietet das *Distack-Framework* ein datenzentriertes Nachrichtensystem zur lokalen Kommunikation zwischen Modulen und zur verteilten Kommunikation zwischen Modulen entfernter *Distack*-Instanzen. Hierzu registrieren sich Module für die Informationen, die sie empfangen möchten. Dies wird durch die Komponenten *MessagingSystem*, *RemoteMessaging* und *Communication* realisiert. Zusätzlich können Module bei der Registrierung angeben, ob sie diese Information auch empfangen möchten, falls sie von einer entfernten *Distack*-Instanz gesendet wurden. Hat ein Modul Informationen, welche es für interessant für andere lokale oder entfernte Module hält, sendet es diese Informationen in das Nachrichtensystem. Dabei kann das Modul angeben, ob die Information nur lokal verteilt oder auch an entfernte *Distack*-Instanzen versendet wird. Zum tatsächlichen Versand an entfernte Instanzen können Mechanismen wie Sockets oder Signalisierungsmechanismen wie z. B. GIST [3] verwendet werden. Durch spezielle Mechanismen zur Serialisierung und Deserialisierung [4] ist es nicht notwendig, neue PDU-Formate zum Transport von Nachrichten an entfernte Instanzen zu definieren. Der Versand und Empfang von lokalen und entfernten Nachrichten ist hierdurch so weit möglich transparent für Module. Ein weiterer Vorteil des verteilten Nachrichtensystems ist die Möglichkeit ein System zur Angriffserkennung, bestehend aus einer Menge von Modulen, auf mehrere Systeme zu verteilen. Dadurch kann Redundanz und somit Ausfallsicherheit erhöht werden sowie Module entsprechend der auf den Systemen zur Verfügung stehenden Ressourcen verteilt werden.

Als letzte Komponente besitzt das *Distack-Framework* Hilfsfunktionalität zur einfachen und transparenten Verwendung von oft benötigter Funktionalität. Beispielsweise ist hier der Zugriff auf die Einträge der lokalen Routingtabelle eines Systems zu nennen. Dieser unterscheidet sich unter Windows, Linux und einem Simulator grundsätzlich. Weiterhin müssen z. B. Timer in einer Simulationsumgebung anders arbeiten um der geänderten und nicht zwingend zeitkontinuierlichen Zeitdomäne gerecht zu werden. *Distack* bietet transparenten Zugriff auf solche Funktionalität. Somit können Module schnell und unkompliziert entwickelt werden.

Das *Distack-Framework* ist derzeit unter Windows, Linux und dem Netzwerksimulator OMNeT++ (siehe auch [5]) lauffähig. Dies bedeutet, dass entwickelte Module transparent in jeder dieser Umgebungen ohne Änderungen des Quellcodes lauffähig sind. Hiermit vereinfacht sich die Entwicklung und Evaluierung von Mechanismen zur Angriffserkennung stark. Weiterhin beinhaltet das Framework bereits eine Vielzahl von Modulen, welche oft benötigte Funktionalität implementieren und direkt verwendet werden könne. Zukünftige Arbeiten werden sich mit der Portierung von *Distack* in neue Umgebungen, wie z. B. programmierbare Netzwerkkarten oder Routerhardware, beschäftigen. Weiterhin werden Mechanismen zur verteilten Angriffserkennung erarbeitet und neue wiederverwendbare Module zur Angriffserkennung und Analyse von Netzwerkverkehr entwickelt.

[1] M. Bairwise, New Zealand teenager accused of controlling botnet of 1.3 million computers, http://www.heise-online.co.uk, Nov. 2007.

[2] T. Gamer, M. Schöller und R. Bless. A Granularity-adaptive System for in-Network Attack Detection. In *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation*, Seite 47–50, Sept. 2006.

[3] Schulzrinne and Hancock. GIST: General Internet Signaling Transport. Internet-Draft, IETF, 2007. Work in Progess.

[4] R. Ramey. Boost Serialization Library, 2002.

[5] C. P. Mayer, T. Gamer, Integrating real world applications into OMNeT++, Institute of Telematics, Universität Karlsruhe (TH), Technischer Bericht, Nr. TM-2008-2, Feb 2008.