„Threat Modeling“
Was kann schon schief gehen?

Zusammenfassung

Bereits während der Entwicklung einer Anwendung sollte man sich mit sicherheitskritischen Aspekten auseinandersetzen. Es darf nicht sein, dass diese während des späteren Betriebs beim Kunden schutzlos zahlreichen Angriffen ausgeliefert ist. Nicht nur vertrauenswürdige Nutzerdaten, sondern auch der Ruf eines ganzen Unternehmens können hierbei auf dem Spiel stehen.

„Threat Modeling“ ist ein systematischer Ansatz zum Entdecken und Dokumentieren von Bedrohungen im Kontext des jeweiligen Anwendungszenarios. Wegweisend ist dabei eine Reduzierung der Angriffswahrscheinlichkeit, indem man identifizierte Schwachstellen in der Software behebt.

Diese Art der Modellierung ist bei bedeutenden Entwicklungsentscheidungen hilfreich und ermöglicht es den Entwicklern, das Design der Anwendung den erforderlichen Sicherheits-richtlinien anzupassen.

1. Grundlagen im „Threat Modeling“

1.1. Terminologie

Asset

Im Zentrum des „Threat Modelings“ stehen die wertvollen Ressourcen. Sämtliche Sicherheitsziele sind darauf ausgelegt, diese „Assets“ zu schützen. So kann die Verfügbarkeit einer Information oder die Information selbst als schutzwürdig eingestuft werden. Andererseits könnten auch weitere immaterielle Aspekte, wie bspw. der Ruf eines Unternehmens, von Bedeutung sein. Im Allgemeinen soll kein nicht autorisierter Zugriff auf vertrauenswürdige Daten möglich sein.

Threat

Die Bedrohung (engl. „threat“) spezifiziert jenes unerwünschte Ereignis, dessen es Beachtung während der Modellierungsphase zu schenken gilt. Die Auswirkungen dieses potentiell möglichen Vorfalls äußern sich im Schädigen oder Gefährden eines „Assets“ bzw. Sicherheitszieles. Ein „Threat“ muss nicht zwingend böswillig sein.

Vulnerability

Eine Sicherheitslücke (engl. „security vulnerability“) bezeichnet die Schwachstelle im System, welche einen Angriff ermöglicht. Hier könnte der Angreifer befähigt werden, schadhaften Code (über ein „Exploit“) auszuführen und „Assets“ gefährden.

Attack

Als Angriff (engl. „attack“) definiert man das Ausnutzen von einer oder mehreren Sicherheitslücken um somit eine Bedrohung für das System hervorzurufen.

Countermeasure

Eine Gegenmaßnahme (engl. „countermeasure“) adressiert Sicherheitslücken um Angriffswahrscheinlichkeiten zu senken. „Threats“ werden hierbei nicht direkt betrachtet, da man sich zwar vor ihnen schützen kann, sie jedoch nach wie vor existieren. Diese Maßnahmen reichen von Designoptimierung über Codeverbesserung bis hin zur Überarbeitung der grundlegenden Programmabläufe.

1.2. Schlüsselkonzepte

Während des „Threat Modelings“ ist es wichtig, grundlegende Ideen dieser Modellierungsart nicht aus den Augen zu verlieren, damit unnötige Unstimmigkeiten im Team vermieden werden können.

Im Wesentlichen ist es erstrebenswert, die Wahrscheinlichkeit von Angriffen auf die Anwendung auf ein Minimum zu reduzieren. Stets müssen Stellen identifiziert werden, welche erhöhte Aufmerksamkeit erfordern. Basierend auf eigenen Erfahrungen und anderen Produkten auf dem Markt sind natürlich zahlreiche Schwachstellen, Bedrohungen und Angriffe, bsp. populärer Frameworks und Webserver, bekannt. Normalerweise finden sich jedoch nicht alle im zu entwickelnden System wieder. Vieles kann, nicht alles muss passieren.

Zeit und Geld beschränken den Aufmerksamkeitsspielraum. Man darf bei der Abwägung der Relevanz von neuen Informationen nie den Anwendungstyp und die zugehörigen Szenarien unbeachtet lassen. Diese **Kontextpräzision** ist essentiell für den Modellierungsvorgang. Unwichtige „Threats“ sollten frühzeitig abgegrenzt und der weitere Ablauf optimiert werden.

Die Vorgehensweise beim „Threat Modeling“ ist inkrementell. Bei jeder neuen **Iteration**sstufe steigt der Detailgrad der Modelle, wenn aktualisierte Fakten berücksichtigt werden. Es ist völlig unproblematisch, falls relevante Informationen „zu spät“ bzw. nach Beendigung eines Durchlaufs bekannt werden. Diese berücksichtigt man einfach in der nächsten Stufe.

Klare Sicherheitsziele sind unerlässlich. Man sollte sich bewusst sein, was unter keinen Umständen im System passieren darf oder was in gewissem Rahmen zulässig ist. Weiterhin muss feststellbar sein, wann die **Qualität** eines Modells genügt, sodass die Vorgabe eines zeitlichen Rahmens möglich wird. Mit Hilfe von Tests kann man diese Kriterien überprüfen.

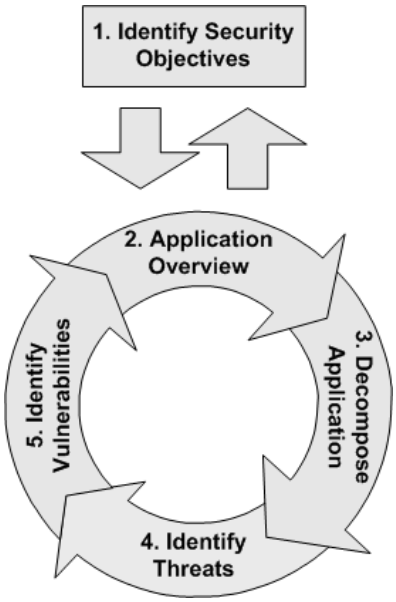
Das Entwicklerteam muss zusammenarbeiten. Gefahren und Sicherheitslücken müssen dokumentiert und verstanden werden. Verteiltes Wissen sollte mittels reibungsfreier **Kommunikation** allen zugänglich sein.

Wiederverwendbarkeit spart Zeit und Geld. Während des Modellierens sollte man Zusammenhänge zwischen Problemen und deren Lösungen erkennen und kategorisieren. Dieses **muster**basierte Informationsmodell reduziert den Bedarf an Expertenwissen, da nicht ständig fundamentale Aufgaben aus dem Nichts heraus neu bearbeitet werden müssen.

2. Modellierungsansatz

1. Identifikation der „Assets“ bzw. Sicherheitsziele

Am Anfang sind normalerweise Anforderungen, Sicherheitsrichtlinien, Normen oder sonstige Vorschriften gegeben, welche auf Schlüsselziele abgebildet werden können. Diese klaren Ziele helfen bei der Aufgabenverteilung und –übersicht. Während der Entwicklungsphase ist es durchaus möglich, dass sich die Anforderungen an die Software ändern. Jedes mal, wenn neue Informationen verfügbar werden, sollte man sie gegen die aktuell erforderlichen Richtlinien abgleichen und das über weitere Vorgehen entscheiden.



1 - Ablauf beim "Threat Modeling"

2. Übersicht der Architektur

Einfache Diagramme und Tabellen ergeben einen groben Überblick über die Softwarearchitektur. Wichtige Anwendungsfälle (engl. „use cases“) und verwendete Technologien bieten Charakteristika, welche helfen, Bedrohungen zu identifizieren.

3. Dekomposition der Architektur

Hier wird die Architektur detailliert untersucht. Bsp. wird bei Webanwendungen das zugrunde liegende Netzwerk und die Infrastruktur des Hosts betrachtet. Es werden Vertrauensgrenzen gesetzt und mittels Datenflussdiagrammen interne Abläufe nachvollziehbar dargestellt. So kann man auch Einstiegspunkte des Nutzers in die Software aufzeigen.

4. Bedrohungen herausstellen

An Hand der Sicherheitsrichtlinien, dem gewonnenem Architekturwissen und allgemein bekannten „Threats“ kann man nun herausfinden, welche Bedrohungen für das System realistisch sind (siehe „STRIDE“-Modell). Danach müssen diese natürlich bewertet werden, damit man jeder relevanten Bedrohung ihren angemessenen Aufmerksamkeitsgrad zuordnen kann (siehe „DREAD“-Modell). Parallel dazu ist eine gute Dokumentation unentbehrlich. Diese Daten könnten in ein Bugtracking-System eingepflegt und direkt dessen Reportfunktionalitäten genutzt werden.

5. Sicherheitslücken finden

Die identifizierten Bedrohungen müssen mit gegebenen Schwachstellen in Verbindung gebracht werden. Auch hier können allgemein gültige Informationen über Schwächen in verwendeten Technologien hilfreich sein, um vorliegende Sicherheitslücken zu finden.

2.1. „STRIDE-Modell“ – Bedrohungen identifizieren

Dieses Modell bildet einen zielbasierten Ansatz zur Identifikation von Bedrohungen [3][7]. Die Ziele des Angreifers stehen im Vordergrund.

- **Spoofing Identiy**

  Der Angreifer täuscht eine falsche (Server-/Client-)Identität vor und kann so auf vertrauliche Daten zugreifen oder diese abfragen. Im einfachsten Fall könnte dies durch die Verwendung fremder „Credentials“ (Benutzername-Passwort-Kombination) geschehen.

  Starke Authentifikation in Verbindung mit Verschlüsselung des Datenkanals oder Mehr-Faktoren-Authentifikation kann dieser Täuschung entgegenwirken.

- **Tampering with Data**

  Der Angreifer manipuliert ohne Autorisierung (persistente) Daten. Dies könnten bsp. Passwortdatenbaken, Preise in einem Online-Shop oder sogar Netzwerkpakete sein. Durch modifizierte Audit-Logs würden möglicherweise Spuren eines Angriffs verwischt.

  Passwörter sollten nicht im Klartext, sondern lediglich ihre Hash-Codes hinterlegt werden. Digitale Signaturen und Verschlüsselung bietet zudem erhöhte Sicherheit vor dieser Bedrohung.

- **Repudiation**

  Der Angreifer leugnet eine Tat. Nachdem ein Angriff identifiziert und sogar die verantwortliche Person gefunden wurde, mangelt es dennoch an Beweisen, den Täter zu überführen.

  Hier sollten die Audit-Logs vor Manipulation geschützt werden. Empfangsbestätigungen, digitale Signaturen und Zeitstempel sind für die Beweisbarkeit sinnvoll.

- **Information Disclosure**

  Der Angreifer gelangt an vertrauliche Daten. Dies könnten lokale Dateien, Informationen über die Infrastruktur eines Webservers, Netzwerkpakete oder sogar Fehlermeldungen sein. Möglicherweise gelangt ein Eindringling über einen nicht autorisierten Datenbankzugriff an sämtliche Nutzerdaten eines Systems.

  Diesbezüglich sind eine starke Authentifikation in Verbindung mit Verschlüsselung und eine durchdachte Zugriffskontrolle (Autorisierung) erforderlich. Im Allgemeinen sollte jedoch ein sensilber Umgang mit vertraulichen Informationen eingehalten werden. Es muss u. U. nicht alles gespeichert werden.

- **Denial of Service**

  Ziel des Angreifers ist es, die Ressourcen des Servers (Rechenzeit, Speicherkapazität, …) aufzubrauchen und so die Verfügbarkeit der angebotenen Dienste zu unterbinden. Diese Angriffe sind sowohl auf Netzwerk- als auch auf Softwareebene möglich. Hier werden u. a. „race conditions“ ausgenutzt, in dem bspw. eine zu hohe Anzahl von eingehenden Request-Paketen, beim Abarbeiten dieser, Fehlverhalten auslöst.

  Gegenwirkend müssten Verfügbarkeit und Zuverlässigkeit von Server und Webanwendung sichergestellt werden. Überwachende Prozesse könnten anormales Verhalten erkennen. Die IP-Adresse des Angreifers sollte vom Zugriff ausgeschlossen werden.

- **Elevation of Privilege**

  Der Angreifer möchte bsp. mittels „buffer overflow“ oder „SQL injection“ seine Rechte auf dem Server oder in der Anwendung erhöhen. Als Endziel werden die Administratorrechte angesehen.

  Ein robuster Code und Eingabevalidierung dienen dem Schutz vor solchen Angriffen. Die Nutzer der Anwendungen sollten zudem nur minimal notwendige Rechte besitzen.

2.2. Bedrohungen bewerten

Sofern man Bedrohungen als relevant (für das System) eingestuft hat, müssen diese bewertet werden [6], damit im weiteren Verlauf ersichtlich ist, welche Problemen dringlich und welche vorerst vernachlässigbar sind. Ein einfacher Ansatz hierfür wäre:

Risiko = Angriffswahrscheinlichkeit * Schadenspotential

Unter der Annahme, dass die Faktoren von 1 bis 10 skalieren, resultieren für das Risiko Werte im Bereich 1 bis 100. Dieser lässt sich in drei Teile „High“, „Medium“ und „Low“ gliedern, damit das Ergebnis leichter zu handhaben ist.

Solch ein Verfahren ist aber meist nicht praktikabel, da die Zahlen schwer von allen Teammitgliedern nachzuvollziehen sind und es wird nötig, erweiterte Sichtweisen auf eine Bedrohung zu definieren:

„DREAD“-Modell – Bedrohungen genauer bewerten

Im Vorfeld muss für das Projekt festgelegt werden, welche Charakteristika bei einer Bedrohung genauer untersucht werden. Diese sollten natürlich gut dokumentiert werden, damit jeder sie auch im Nachhinein verstehen kann.

Im Folgenden seien nun die grundlegenden Kriterien aufgezeigt, welche je nach Domäne ergänzt werden können:

- **Damage Potential → Welcher Schaden kann angerichtet werden?**
  1) Unwichtige Daten werden bekannt.
  2) Wichtige Daten werden bekannt.
  3) Der Angreifer erlangt Administratorrechte.

- **Reproducibility** → **Ist der Vorgang reproduzierbar?**
  1) Selbst mit Kenntnis der Schwachstelle ist es schwer zu wiederholen.
  2) Nur mittels Zeitfenster und „race conditions“ wiederholbar.
  3) Es ist stets reproduzierbar.
- **Exploitability** → **Wie wahrscheinlich ist ein Angriff?**
  1) Nur ein sehr erfahrener Programmierer kann den Angriff verüben.
  2) Mit einem Werkzeug ist es durchführbar.
  3) Selbst dem Laien ist es (unbewusst) möglich.
- **Affected Users** → **Wer ist betroffen?**
  1) Kein Benutzer ist betroffen.
  2) Nur ein Benutzer ist betroffen.
  3) Eine ganze Benutzergruppe ist betroffen.
- **Discoverability** → **Wie schnell kann es bemerkt werden?**
  1) Bei normalem Gebrauch der Software fast nicht zu entdecken.
  2) Dem Nutzer könnte es auffallen.
  3) Der Angriff ist direkt aus den Protokolldaten ersichtlich.

Desweiteren könnte man sich noch die Frage der „**Reputation**“ (dt. „guter Ruf“) stellen:

- Was steht auf dem Spiel?
- Besteht die Gefahr, das Vertrauen des Nutzers zu verlieren?

Jeder Frage werden 1-3 Punkte zugeordnet, sodass im Basisfall 5-15 Punkte pro Bedrohung zu vergeben sind. Damit dieses Ergebnis noch deutlicher wird, kann man es wieder in drei Teile aufspalten: „Low“ 5-7, „Medium“ 8-11, „High“ 12-15.

2.3. Ergebnisse

„Threat Modeling“ ist iterativ! Dieser Modellierungsprozess ist nicht nach den 5 beschriebenen Schritten abgeschlossen. Es müssen jederzeit neue Aspekte berücksichtigt und in die vorhandenen Modelle eingebracht werden.

Nach jedem Iterationsschritt erhält man detailierte Informationen über die gegebene Anwendungsarchitektur und die zugehörigen Sicherheitsaspekte, welche es zu beachten gilt. Desweiteren hat man nun eine Liste mit bewerteten, bedeutsamen Bedrohungen und im günstigen Fall wurden bereits jetzt Schwachstellen im System identifiziert. Die **Designer** des Entwicklerteams können diese Daten nun als Hilfestellung für weitere Entwurfsentscheidungen nutzen, wenn es um das Verwenden von neuen Technologien oder sonstigen Funktionalitäten geht. **Programmierer** sollten jetzt in der Lage sein, gezielt im Code Risiken abschwächen zu können und mit **Tests** ganz bewusst Angriffe durchgespielen können, um zu überprüfen, ob analysierte Anfälligkeiten weiterhin gegeben sind.

Nur die Risiken eines Angriffs können abgeschwächt werden. Identifizierte Bedrohungen können weder gemäßigt, noch eliminiert werden, da sie unabhängig von den ergriffenen Sicherheitsmaßnahmen existieren. Für die weitere Entwicklung sind sie lediglich uninteressant geworden, wenn die betreffenden Sicherheitslücken geschlossen wurden.

3. Web Application Security Frame

Im Hinblick auf die Entwicklung von Webanwendungen gibt es zahlreiche Bereiche, in denen häufig Fehler bezüglich der Sicherheit gemacht werden und damit erhöhte Aufmerksamkeit erfordern [5]:

- Validation von (Eingabe-)Daten
- Authentifizierung
- Autorisierung
- Konfigurationsmanagement
- Umgang mit vertrauenswürdigen Daten
- Session-Management
- Kryptografie
- Parametermanipulation
- Ausnahmebehandlung
- Protokollierung

Im Folgenden seien nun die Kategorien herausgegriffen, welche auch für die Konfiguration von Webservern wichtig sind.

Authentifizierung

Im Schritt der „Authentifizierung“ beweist eine Entität die Echtheit einer Identität einer anderen Entität. Dies geschieht meist mittels Benutzername und Passwort. So stellt der Server gegenüber dem Client die Frage „Wer bist du?“, welcher diese daraufhin wahrheitsgemäß beantworten muss.

Problematisch sind hier schwache Kennwortrichtlinien und unsichere Authentifizierungs-methoden, wie Basisauthentifizierung. So können kurze und einfache Passwörter (ohne Sonderzeichen, bekannte Wörter enthaltend) mittels „dictionary attacks“ oder „brute force attacks“ mit geringem Zeitaufwand erspäht werden. Sollten die Kommunikationskanäle nicht verschlüsselt sein, genügen schon einfache Netzwerküberwachungstools um die Datenpakete abzufangen und zu lesen.

Autorisierung

Die Zugriffskontrolle für Ressourcen auf dem Webserver behandelt die Frage: „Was darfst du tun?“. Ein durchdachtes Rechtesystem kann viele vertrauenswürdige Daten vor nicht autorisierter Abfrage schützen. An dieser Stelle sei auf die im „STRIDE“-Modell angesprochenen Punkte „Elevation of Privilege“, „Information Disclosure“ und „Tampering with Data“ verwiesen.

Protokollierung

Dieser Aspekt beschäftigt sich mit der Aufzeichnung aller sicherheitsrelevanten Ereignisse. Fehlerquellen finden sich hier bspw. beim Ignorieren fehlgeschlagener Logins oder unzureichender Absicherung der Audit-Logs. Ein Angreifer sollte nicht die Möglichkeit besitzen, seine Spuren durch Manipulation der Logfiles zu verwischen (siehe „Repudiation“). Je mehr man aufzeichnet, desto mehr kann man aus den Protokollen erkennen. Hier ist jedoch der datenschutzrechtliche Gesichtspunkt zu beachten. Nutzer dürfen eventuell nicht während ihrer kompletten Arbeitsweise mit der Anwendung überwacht werden. So könnte es im Falle eines Angriffs passieren, dass der Administrator das Beweismaterial, welches den Angreifer überführen würde, nicht verwenden darf.

3.1. Authentifikation und Autorisierung im IIS

Im Folgenden werden einige Möglichkeiten der Authentifikation in den Microsoft Internet Information Services (IIS) vorgestellt. Diese sind, teilweise in leicht abgewandelter Form, auch mit dem Apache HTTPD nutzbar. Zudem wird auf einige Aspekte der Zugriffskontrolle eingegangen.

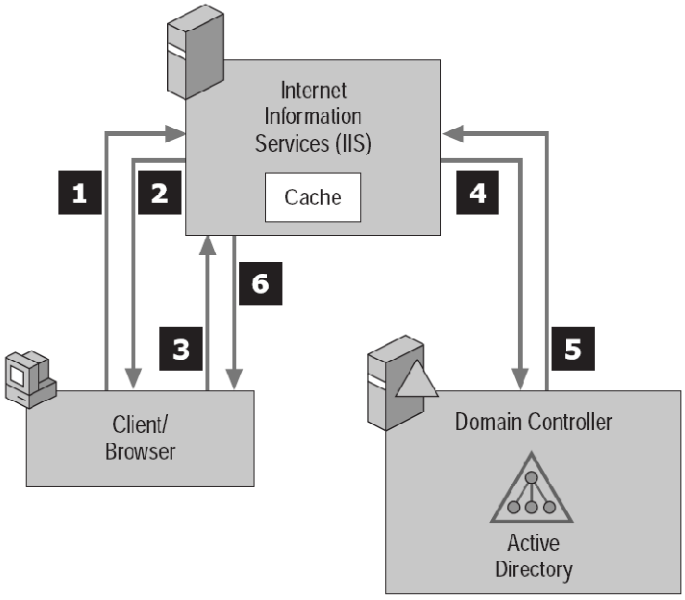
Anonyme Authentifizierung

Bei dieser Methode ist es nicht nötig, dass der Client seine Identität beweist, da er als „allgemeiner Gast“ auf dem Server akzeptiert wird und auf alle öffentlichen Ressourcen zugreifen darf. Dieser Zugriff kann noch mittels IP-Adressmasken eingeschränkt werden.

Basisauthentifizierung

Hier wird der Client nach Benutzername und Passwort gefragt, welche sich im Falle vom IIS mit den zulässigen Kontodaten aus der SAM-Datenbank („security account manager“) [11] von Windows decken müssen. Natürlich muss dieser Benutzer für die entsprechende Ressource zugriffsberechtigt sein, was direkt über die „Access Control List“ (ACL) [10] aus dem Dateisystem geregelt wird. Die Übermittlung der „Credentials“ erfolgt im übertragungsfreundlichen Base64-Format – also unverschlüsselt. Sollte der Kommunikationskanal nicht mit SSL o. ä. abgesichert sein, stellt diese Authentifizierungs-möglichkeit ein hohes Sicherheitsrisiko für die Webanwendung dar.

Digest-Authentifizierung



2 - Ablauf der Digest-Authentifizierung

Wenn der Client eine Datei anfordert (1), verweigert der Server initial den Zugriff darauf und sendet, dass „Digest-Authentifizierung“ genutzt wird und wie der Bereichsname („realm“) lautet (2). Daraufhin fragt der Browser des Clients Benutzername/Passwort ab. Diese Daten werden im MD5-Hash mittels wiederholter Anfrage im HTTP-Header zum Server gesendet (3). Der Benutzername ist dem Server als Klartext bekannt. Der Client-Hash wird nun zum Domain Controller (DC) gesendet (4), welcher daraufhin die Daten prüft und das Resultat dem Webserver mitteilt (5,6). Damit dieser Vorgang funktioniert, müssen die Passwörter im Active Directory des DC reversibel verschlüsselt sein, da der DC selbst auch den Hashwert zum Vergleich nachbilden muss. Bei unzureichender Absicherung des DC könnte dies ein Sicherheitsrisiko darstellen.

Abhilfe schafft die **erweiterte Digest-Authentifizierung** [12], bei der die Client-Hash-Daten als extra Feld am Benutzerobjekt im Active Directory liegen. In diesem Fall muss natürlich der DC den „Realm“ vorgeben, da dieser in den Hashwert mit einfließt. Dieser Unterschied wird nur beim IIS gemacht, die allgemeine „Digest-Authentifizierung“ [13] adressiert jene „erweiterte“ Variante.

3.2. CLF – Common Log File Format

Das CLF ist ein standardisiertes Textdateiformat [2], welches von Webservern genutzt werden kann um Logfiles zu erzeugen. Dank des Standards können die resultierenden Daten von diversen Programmen analysiert werden. Jede Zeile setzt sich nach folgendem Schema zusammen:

```
host ident authuser date request status bytes
```

So wird der DNS-Hostname oder die IP-Adresse des Anfragenden, dessen Loginname, Datum und Uhrzeit, den exakten Wortlaut der Anfrage des Clients, der resultierende HTTP-Status-Code und der Umfang der übertragenen Daten aufgezeichnet.

Auf Grund des geringen Informationsgehalts exisitert auch ein erweiterter Standard für diese Protokollierung: „W3C Extended Log File Format“ (ELF) [9]. Dieser ist in der Lage den erhöhten Transaktionsdatenspielraum verschiedener Webserver aufzuzeichnen.

Um die riesigen Datenmengen der Audit-Logs auch auswerten zu können, sei hier beispielhaft das Programm „Webalizer“ [8] aufgeführt. Es verarbeitet nach eigenen Angaben auf einem 200-MHz-PC bis zu 10.000 Einträge pro Sekunde. Es wurde vollständig in C geschrieben und steht unter der GPL v2. Normalerweise akzeptiert es CLF-Dateien, aber nach Konvertierung sind auch ELF-Dateien auswertbar.

4. Quellen

[1] Threat Modeling Web Applications
(http://msdn2.microsoft.com/en-us/library/ms978516.aspx)

[2] Common Log File Format
(http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format)

[3] STRIDE-Threat Modeling
(http://kennymaita.blogspot.com/2007/02/s-t-r-i-d-e-threat-modeling.html)

[4] Threat Modeling – Casaba Security
(http://casabasecurity.com/threat_modeling)

[5] Cheat Sheet: Web Application Security Frame
(http://msdn2.microsoft.com/en-us/library/ms978518.aspx)

[6] Threat Modeling & DREAD-Model
(http://msdn2.microsoft.com/en-us/library/aa302419.aspx)

[7] STRIDE-Model
(http://www.leastprivilege.com/content/binary/Threat%20Modeling%20und%20sicherer%20Software-Entwicklungszyklus.pdf)

[8] Webalizer
(http://www.mrunix.net/webalizer)

[9] Extended Log File Format
(http://www.w3.org/TR/WD-logfile.html)

[10] Access Control List
(http://en.wikipedia.org/wiki/Access_control_list)

[11] Security Account Manager (Windows NT)
(http://en.wikipedia.org/wiki/Security_Account_Manager)

[12] IIS Resource Kit – TechNet
(http://www.microsoft.com/downloads/details.aspx?FamilyID=80a1b6e6-829e-49b7-8c02-333d9c148e69)

[13] Digest-Authentifizierung
(http://en.wikipedia.org/wiki/Digest_access_authentication)