Implementation und Review einer "Home-Grown Certification Authority"

Roman Racine, Markus Heule, Stefan Birrer, David Fuchs

13. Januar 2005

Implementation einer Certification Authority

Der Aufbau unserer Certification Authority besteht aus 2 Rechnern. Die benötigte Funktionalität (Erstellen und Revozieren von Zertifikaten, Datenbankverwaltung, User- und Admin-Schnittstelle) werden dabei vom Server "Cauth" übernommen. Der zweite Rechner ("Cauth_Backup") wird zur Speicherung von Backups verwendet. Für den Zugriff via Web sind auf Cauth ein Apache-Webserver sowie einige benötigte Apache-Modules installiert. Daneben steht Perl als Standardinstallation mit den Zusatzmodulen DBI und DBD::mysql zur Verfügung. Als Datenbank-Server verwenden wir MySQL. Die selbst programmierte Funktionalität des Systems besteht aus einigen Perl-Modulen mit mehrfach benötigtem Code sowie den Frontends für den Web-Zugriff für die Administratoren und Benutzer. Die Perl-Module enthalten im Wesentlichen Code für die Datenbankabfragen und HTML-Präsentation der Resultate. Weiter ist auch die Schnittstelle zu OpenSSL (Erstellung von Zertifikaten) in Perl programmiert.

Aufbau des Systems

Betriebssystem

Als Operating System für den Server verwenden wir ein Gentoo-System mit aktuellem 2.6-Kernel. Das System wurde so minimal wie möglich konfiguriert, um die benötigte Funktionalität zu erreichen und dabei das von nicht benötigten Anwendungen ausgehende potentielle Sicherheitsrisiko (etwa durch ungewolltes Herstellen von TCP-Verbindungen) zu minimieren. Als einziger Benutzer-Account existiert der root-Account sowie Systemaccounts beispielsweise für den Webserver. Wir haben versucht, die Schranken für das unerlaubte Erlangen von lokalen Benutzer-Rechten so hoch als möglich zu legen. Auf das Verhindern von Attacken durch Angreifer, welche dennoch solche Rechte erlangen konnten, haben wir kein grosses Augenmerk gelegt. Durch Anlegen zusätzlicher Schranken zur Verhinderung unerlaubter Aktivitäten eines lokalen Benutzers könnte die Sicherheit des Systems noch erhöht werden. Aus Zeitgründen haben wir ebenfalls darauf verzichtet, sämtliche installierten Pakete auf den aktuellsten Stand zu bringen. Vor einem produktiven Einsatz des Systems hätte dieser Schritt unbedingt noch durchgeführt werden müssen.

Netzwerk-Setup

Auf Cauth sind Sämtliche nicht benötigten Netzwerk-Ports geschlossen. Offen sind lediglich der HTTPS-Port für Verbindungen zum Web-Server sowie der SSH-Port zur Remote Administration. Zusätzlich ist eine Statefull Firewall aktiv, die unerwünschte eingehende Verbindungen unterbindet. Diese Kombination schützt beispielsweise vor externen Zugriffen auf unbemerkt laufende Dienste. Wir erachten das System allerdings in der aktuellen Konfiguration auch als genügend geschützt, sollte die Firewall einmal ihren Dienst versagen. Ausgehende Verbindungen sind aus praktischen Gründen erlaubt. Sie erleichtern die Administration des Systems erheblich. Ein Sperren solcher Verbindungen würde es einem durch einen Angreifer installierten "Trojaner" signifikant erschweren, Daten auf das System des Angreifers zu transferieren. Wir haben die Gefahr eines erfolgreichen Installationsversuchs eines solchen "Trojaners" allerdings als sehr klein eingeschätzt. Daher überwogen hier die Vorteile für die Benutzbarkeit des Systems. In einem produktiven System, welches selten administriert werden muss, könnte diese zusätzliche Sicherheitsmassnahme allerdings Sinn machen. Durch die ausschliessliche Verwendung von verschlüsselten Verbindungen erachten wir das Risiko eines Hijackings von TCP-Verbindungen als äusserst gering. Wir haben daher auf Veränderungen am Kernel bezüglich Netzwerk-Optionen (etwa die Wahl von zufälligen Sequenznummern) verzichtet. Der Datenbank-Server kann nur via UNIX-Sockets angesprochen werden. Zugriffe auf Datenbank-Informationen sind damit auf den lokalen Host beschränkt.

Web-Server

Als Webserver wird Apache in der aktuellen Version 2.0.52 eingesetzt. Die Benutzer-Authentisierung wird vollständig von Apache übernommen. Dabei werden bei Logins mit Zertifikat die Zertifikate geprüft und mit der Certificate Revocation List verglichen. Bei Logins mit Benutzername und Passwort wird mithilfe der MySQL-Datenbank geprüft, ob die Login-Daten korrekt sind. Für die Authentisierung benutzen wir keine selbst programmierte Funktionalität.

Folgende Apache-Module werden von uns eingesetzt:

- *mod_ssl:* Stellt die erforderlichen kryptographischen Funktionen für die Verwendung von SSL-verbindungen zur Verfügung.
- *mod_auth_mysql:* Erlaubt den Schutz von Web-Zugriffen auf Seiten durch MySQL-verwaltete Benutzerinformation.
- *mod_perl:* Apache-Interface für die Ausführung von PERL-scripts. Weitere Module werden nicht benötigt und wurden auch nicht installiert.

Es werden ausschliesslich SSL-Verbindungen auf Port 443 (HTTPS) akzeptiert. Folgende Server-Verzeichnisse sind via Web erreichbar:

- */cgi-bin/admin:* Dieses Verzeichnis enthält das Administrator-Interface. Um darauf zugreifen zu können, muss der Benutzer ein gültiges Zertifikat vorweisen, das von unserer CA signiert ist und als "Organizational Unit" des Users den Wert *admin* aufführt. Mit diesem Eintrag wird verhindert, dass beliebige Benutzer Zugriff auf das Administrator-Interface haben.

- */cgi-bin/usercert:* Dieses Verzeichnis erlaubt den Zugriff auf das User-Interface mittels Zertifkatsauthentisierung. Das Zertifikat muss von unserer CA signiert sein.
- */cgi-bin/userpw:* Dieses Verzeichnis ist für den Zugriff mittels Passwort gedacht. Es wird mod_auth_mysql verwendet, um das Passwort direkt in der Imovies Datenbank zu überpruefen.

Die Verzeichnisse /cgi-bin/usercert und /cgi-bin/userpw sind symbolische Links auf das Verzeichnis /cgi-bin/user. Die Unterscheidung muss nur für die verschiedenen Authentisierungsmethoden in Apache gemacht werden.

Datenbank

Die MySQL-Datenbank enthält neben der vorgegebenen Relation *users* die Relationen *admins* (welche die *uid*s aller Personen enthält, die Administrator-Rechte besitzen) und *certs*, welche die Liste der aktuellen Zertifikate und der dazugehörigen Benutzer enthält. Die *revoked*-Relation enthält Information über widerrufene Zertifikate.

Admin-Schnittstelle

Folgende Informationen sind über die Admin-Schnittstelle abrufbar:

- Der Inhalt der obengenannten Tabellen.
- Ein Überblick über die Anzahl Benutzer, die Anzahl ausgestellter Zertifikate, die Anzahl revozierter Zertifikate und die aktuelle Zertifikats-Seriennummer. Der Zugriff auf die Admin-Schnittstelle ist nur mit einem Zertifikat möglich, das als "Organisational Unit" (OU) den String *admin* enthält.

User-Schnittstelle

Mithilfe der User-Schnittstelle können Benutzer

- Ein Zertifikat ausstellen. Dabei wird dem Benutzer zunächst die Möglichkeit gegeben, Name, E-Mail-Adresse und Passwort zu aktualisieren. Danach kann das Zertifikat heruntergeladen werden. Ein eventuell schon vorhandenes Zertifikat wird dabei automatisch revoziert.
- Ein Zertifikat widerrufen. Ein widerrufenes Zertifikat wird sofort ungültig.

Der Zugriff auf die User-Schnittstelle kann entweder über ein gültiges Zertifikat oder über gültige Login-Daten erfolgen. Benutzer, die gemäss Datenbank Administrator-Rechte besitzen, erhalten automatisch ein Administrator-Zertifikat, wenn sie sich ein Zertifikat ausstellen lassen.

Backup

Für das Backup von Benutzer-Daten und Zertifikaten haben wir uns für den Einsatz eines separaten Rechners entschieden. Dies erhöht einerseits die Sicherheit der Lösung gegen Angriffe: Gelingt es einem Angreifer nicht, beide Rechner unter seine Kontrolle zu bringen, so entsteht bei einem Angriff zumindest kein Datenverlust. Andererseits sind die Daten so auch bei Hardwareschäden besser geschützt. Natürlich muss bei der Implementierung der Lösung darauf geachtet werden, dass durch die notwendigen Kommunikationsmöglichkeiten beider Systeme keine neuen Sicherheits-Schwachstellen geschaffen werden. Als Backup-Rechner verwenden wir ein im Wesentlichen zum Server identisches System. Die Konfiguration unterscheidet sich in folgenden Punkten:

- MySQL-Datenbankserver und Apache-Webserver werden auf Cauth_Backup nicht benötigt und wurden deshalb hier nicht installiert.
- Nur eingehende SSH-Verbindungen werden akzeptiert, mit den untengenannten Einschränkungen.

SSH ist so konfiguriert, dass nur von der IP-Adresse von Cauth ausgehende Verbindungen akzeptiert werden. Da IP-Adressen gespooft werden können, wird weiter die Authentisierung des verbindenden Rechners über ein Zertifikat verlangt. Die Authentisierung mit Benutzername und Passwort ist nicht möglich. Durch das Zertifikat von Cauth_Backup wird sichergestellt, dass das Backup auch wirklich an den richtigen Rechner gesendet wird. Wird die Adresse von Cauth_Backup gespooft, so wird damit zwar das Backup verhindert, dies wird allerdings bemerkt und ein entsprechender log-Eintrag erstellt. Zusätzlich ist SSH mit einem "forced command" konfiguriert. Dieses ruft ein Shell-Skript auf, welches sicherstellt, dass nur ein RSync-Kommando für ein bestimmtes Directory ausgeführt werden kann.

Das Backup-Skript auf Cauth archiviert einen Dump der gesamten Datenbank auf Cauth_Backup. Dieses Skript wird bei jedem Erstellen oder Revozieren eines Zertifikates ausgeführt. Damit keine für Angreifer nützliche Information den Server verlässt, werden die Backups mittels gpg verschlüsselt, bevor Sie an den Backup-Rechner gesendet werden. Dies geschieht mit der grösstmöglichen Schlüssellänge, und der zur Entschlüsselung benötigte Private Key ist auf keinem der beiden Rechner gespeichert. Nach der Verschlüsselung wird das Backup über eine SSH-getunnelte RSync-Verbindung auf den Backup-Rechner übertragen. Ein Shell-Skript versieht das Backup danach mit einem timestamp und verschiebt es in ein permanentes Unterverzerzeichnis des root-Verzeichnisses.

Eine Entschlüsselung des Netzwerkverkehrs zwischen Cauth und Cauth_Backup durch einen Angreifer halten wir bei der eingesetzten Verschlüsselung für unmöglich. Zu beachten ist, dass die zweifache Verschlüsselung (gpg sowie SSH) wahrscheinlich keinen Sicherheitsgewinn bringt. Durch den Einsatz beider Verfahren ist aber zum einen die Authentizität der Verbindung für beide Rechner sichergestellt. Zudem erhält ein Angreifer, der den Backup-Rechner unter seine Kontrolle bringen kann, keine sensiblen Daten.

Sicherheitsanalyse

Im Bezug auf die serverseitige Programmierung haben wir folgende mögliche Sicherheitsprobleme berücksichtigt:

1. Programmierfehler
2. Angriffe auf das System (Manipulierte Eingaben, Brute Force Attacken, DDoS-Attacken)
3. Fehler von Anwendern (beispielsweise aus Unvorsichtigkeit oder Unwissen)

1. Programmierfehler

Unter Programmierfehlern zusammengefasst sind einerseits fehlerhafte Algorithmen, andererseits auch Fehler, die durch unsaubere Programmierung entstehen (beispielsweise Buffer Overflows). Klassische Sicherheitslücken wie Buffer Overflows können durch die Verwendung einer Programmiersprache mit automatisch verwaltetem Speicher vermieden werden, die dadurch entstehende Einbusse an Effizienz ist vernachlässigbar. Die Verwendung des *strict*-Modus von Perl sorgt ausserdem dafür, dass für Perl typische Fehler wie falsch buchstabierte Variablennamen und mehrfach deklarierte Variablen meist sofort erkannt werden.

Sicherheitslücken, die durch fehlerhafte Algorithmen entstehen, können durch die Verwendung bereits existierender Module (unter Perl z.B. im Comprehensive Perl Archive Network CPAN[1]) minimiert werden, da diese häufig eingesetzten Komponenten bereits seit langem dem Test der Öffentlichkeit standgehalten haben. Konkret haben wir für unser Projekt die DBI und DBD::mysql Module eingesetzt. Im weiteren haben wir uns weitgehend auf die durch Apache und MySQL zur Verfuegung gestellte Funktionalität verlassen, da wir auch hier davon ausgehen, dass breit eingesetzte Standardsoftware punkto Sicherheit ein sehr hohes Niveau aufweist. Aus diesem Grund wird die ganze Überprüfung der Zugriffsberechtigung nicht in unseren selbst programmierten Skripten, sondern ausschliesslich durch den Apache-Webserver vorgenommen, der beim zertifikatsbasierten Zugriff die Gültigkeit des Zertifikats überprüft, beim passwortbasierten Zugriff die Gültigkeit des Passworts direkt anhand der in der Aufgabenstellung vorgegebenen Datenbank überprüft (mithilfe des Apache-Modules mod_auth_mysql). Die Verlagerung der Funktionalität auf den Apache-Webserver hat den Vorteil, dass bei auftretenden Sicherheitslücken meistens rasch Bugfixes verbreitet werden, die bei einem Systemupdate installiert werden, wärend selbst programmierte Komponenten nur dann verbessert werden, wenn der Administrator etwas unternimmt.

Etwas unschön ist die eigene Implementierung der Kommunikation mit OpenSSL, die darauf zurückzuführen ist, dass existierende Perl-Module für OpenSSL leider nicht die für die Aufgabenstellung benötigte Funktionalität anbieten. Die hier auftauchenden Probleme und das eigenwillige Handling von OpenSSL haben scheinbar auch andere Teams vor ähnliche Schwierigkeiten gestellt.

Eine andere Unschönheit betrifft die Impelmentierung der Certificate Revocation List im Apache-Webserver, die das Senden eines Restarts an den Webservers notwendig macht, wenn die CRL verändert wurde. Dies ist leider nicht zu vermeiden.

Zusammenfassend haben wir versucht, durch die Verwendung existierender Standardkomponenten das Risiko von Programmier- und Implementationsfehlern zu minimieren und haben zu demselben Zweck eine Programmiersprache mit automatisch verwaltetem Speicher verwendet und alle Warnungen angeschaltet, die die verwendete Programmiersprache Perl bietet.

[1] http://www.cpan.org/

2. Angriffe auf das System

Wir haben uns Abwehrmassnahmen zu folgenden möglichen Angriffsszenarien überlegt:

- Manipulierte Eingaben
- Brute-Force-Attacken
- Ausnutzen von "Leaking Information"

Manipulierte Eingaben. Manipulationen können einerseits vor, andererseits nach erfolgter Authentifikation auftreten. Die Benutzerführung ist so gehalten, dass die einzig mögliche Manipulation vor der Authentifikation die Übergabe eines manipulierten Zertifikates oder manipulierter Logindaten ist. In diesem Bereich verlassen wir uns voll auf die Funktionalität des Apache-Webservers, der die Authentifikation vollständig übernimmt. Wenn die Daten den serverseitigen Skripten übergeben werden, können wir also davon ausgehen, dass mit der Authentifikation alles in Ordnung ist.

Es verbleibt die Möglichkeit, dass ein angemeldeter Benutzer Formulareingaben manipuliert. Dabei gilt es zwei Fälle zu unterscheiden:

1. Die Formulareingaben ergeben im Kontext überhaupt keinen Sinn oder bewirken, dass die Applikation nicht richtig funktioniert.
2. Durch manipulierte Formulareingaben kann ein Benutzer Dinge tun, die er eigentlich nicht tun können sollte. *(Privilege escalation)*

Zu 1: Da in der Aufgabenstellung keine Gültigkeitsbereiche für die Daten angegeben sind, haben wir selbst unserer Ansicht nach sinnvolle Annahmen getroffen:

a. Die Benutzer-ID ist fix vorgegeben und kann nicht geändert werden.

b. Das Passwort kann beliebige Zeichen beinhalten.

c. Die übrigen Eingabefelder sind auf alphanumerische Zeichen sowie einige andere übliche Zeichen wie zum Beispiel `"."`, `"-"` und `"@"` eingeschränkt. Dies ist für die Sicherheit der Applikation eigentlich nicht notwendig, verhindert aber zum Beispiel, dass ein Benutzer HTML-Tags eingeben kann, die danach bei der Darstellung gesondert behandelt werden müssten.

Bei **b.** und **c.** könnte man leicht andere Annahmen treffen, der entsprechende Code in unserem Skript kann leicht angepasst werden.

Zu 2: Um zu verhindern, dass ein Benutzer mit manipulierten Formulareingaben Dinge tun kann, die er nicht dürfte (beispielsweise durch SQL Injections), verwenden wir konsequent die Möglichkeit des DBI-Moduls, Platzhaltervariablen in SQL-Statements zu verwenden und die Werte getrennt zu übergeben. An Orten, wo dies nicht möglich ist, verwenden wir eine Positivliste aller zulässigen Eingaben und vergleichen die Eingabe damit. Ausserdem verwenden wir konsequent den Taint-Modus von Perl, um eventuelle Fehler bei der Behandlung mit externen Variablen zu erkennen. Externe Eingaben dienen darüber hinaus nie als Parameter für aufgerufene Programme.

Brute Force Attacken. Mögliche Brute-Force Attacken, die wir erkannt haben sind:

a. Unzugänglich machen des Dienstes durch Überschwemmen mit Anfragen

b. Raten von Zugangsdaten durch systematisches Ausprobieren

c. Provokation von Race Conditions

Zu **a**: Grundsätzlich wird ein Dienst immer irgendwann unzugänglich, wenn er mit zuviel Anfragen überschwemmt wird. Allerdings haben wir dafür gesorgt, dass rechenintensive Tätigkeiten wie das Ausstellen von Zertifikaten oder schreibender Zugriff auf die Datenbank ausschliesslich von bereits angemeldeten Benutzern durchgeführt werden kann. Effektiv ist es denkbar, dass ein angemeldeter Benutzer das System lahmlegt, beispielsweise indem er ständig neue Zertifikate anfordert. Immerhin ist ein angemeldeter Benutzer identifizierbar und kann von einem Administrator bei Fehlverhalten ausgesperrt werden.

Zu **b**: Es gibt zwei Möglichkeiten zu unterscheiden:

- Raten eines gültigen Zertifikates: Dies ist nach heutigen Erkenntnissen nicht möglich.
- Raten von Logindaten/Passwörtern: Dies ist durchaus möglich, die Verantwortung, dass dies nicht passiert, legen wir in die Hände des Benutzers, der ein geeignetes Passwort wählen muss.

Zu **c**. Die Provokation von Race Conditions verhindern wir mit folgenden Massnahmen: Einsatz des Tabellentyps *"InnoDB"*. Bei der Verwendung dieses Tabellentyps ist MySQL in der Lage, Transaktionsbasierte Zugriffe auf die Datenbank vorzunehmen, d.h. es gelten die "ACID"-Eigenschaften. Das Auftreten von Race-Conditions im Zusammenhang mit dem Datenbankzugriff sollte so vermieden werden. Veränderte Daten werden erst nach einem expliziten Commit übernommen. Das Ausstellen von Zertifikaten (ein Prozess, bei dem mehrere Programme aufgerufen werden müssen) serialisieren wir mithilfe von Advisory-Locking unter Linux. Da alle zugreifenden Programme Advisory-Locking benützen, ist es nicht möglich, dass mehrere Prozesse gleichzeitig an Zertifikaten manipulieren. Somit sind Race-Conditions ausgeschlossen. Beim Kopieren der neuen CRL in ihr Stammverzeichnis verwenden wir den System Call rename. Bei diesem ist unter Linux und Ext3FS die Atomizität garantiert, folglich kann auch hier keine Race Condition auftreten.

Leaking Information. Fehlermeldungen werden grundsätzlich ins Logfile geschrieben, die Programme sind so geschrieben, dass für eine mit Perl bewanderte Person der Fehler leicht zu lokalisieren sein sollte. Die Information ist jedoch für den normalen Benutzer nicht sichtbar, was allerdings auch Abstriche in der Benutzerfreundlichkeit bedingt, da ein Benutzer unter Umständen nicht weiss, was schief gelaufen ist und ob es an ihm oder am System liegt.

3. Fehlverhalten eines Anwenders

Unter Fehlern, die der Anwender begeht, verstehen wir Sicherheitslücken, die entstehen, weil der Anwender unerfahren ist oder unvorsichtig handelt. Diesen Sicherheitslücken ist gemein, dass sie nur die Privatsphäre des einzelnen Benutzers betreffen, nicht jedoch die Sicherheit des Dienstes oder gesamten Systems. Angesichts der Tatsache, dass die wenigsten Benutzer sich mit der Sicherheitsproblematik auskennen, scheint es jedoch angemessen, sich Gedanken darüber zu machen.

Wir gehen davon aus, dass ein Benutzer grundsätzlich weiss, dass kurze Passwörter leicht zu brechen sind und bei der Wahl eines schwachen Passworts daher das Risiko wissentlich in Kauf nimmt. Dagegen gehen wir nicht davon aus, dass ein Benutzer sich beispielsweise immer ordungsgemäss abmeldet, sondern vielleicht den Browser einfach schliesst. Ein Risiko bildet dabei die mögliche Übernahme fremder Sessions durch jemanden, der nach dem legitimen Benutzer an denselben PC gelangt. Wir haben versucht, hier Abhilfe zu schaffen, indem der Browser sich bei jedem Zugriff neu authentifizieren muss und keine Sessionverwaltung mittels Session-ID, Cookies oder Ähnlichem durchgeführt wird. Nach dem schliessen des Browsers muss der Benutzer das Passwort zur Freischaltung seines Zertifikats oder die Logindaten neu angeben. Andererseits kann ein Dritter, der den noch offenen Browser vorfindet, via Back-Button Zugriff auf die Applikation erhalten. Hier könnte man in Zukunft möglicherweise Abhilfe schaffen, indem man eine Sessionverwaltung einführt und nach einer Zeit von beispielsweise fünf Minuten eine Neuautorisierung verlangt. Wie dies mit Zertifikaten gehen soll, ist jedoch unklar.

Ebenfalls haben wir uns dazu entschieden, dass ein Benutzer jeweils nur ein gültiges Zertifikat haben kann. Dadurch werden Probleme mit vergessenen oder verlorenen Zertifikaten vermieden.

Sicherheitsmassnahmen

Zusammenfassend sollten vor einem produktiven Einsatz unseres Systems folgende Vorkehrungen zur Erhöhung der Sicherheit getroffen werden:

- Sollte ein Angreifer die Rechte eines lokalen Benutzers erlangen, könnten zusätzliche Sicherheitsschranken die Systemsicherheit erhöhen. Im Speziellen sollten die File Access Permissions, etwa der von Apache ausgeführten Skripts, restriktiver gesetzt werden. Die Datenbank könnte durch ein Passwort geschützt werden.
  Eine weitere mögliche Verbesserung würde sich daraus ergeben, den Zugriff von Apache auf die Datenbank einzuschränken. Konkret könnte man MySQL so konfigurieren, dass der Zugriff auf ein Zertifikat nur mit Eingabe eines Benutzerpasswortes möglich ist. Dieses Passwort muss der Benutzer über das Web-Frontend eingeben. Damit wäre es einem Angreifer, der sich die Rechte des Apache-Accounts verschaffen kann, nicht möglich, Zertifikatsdaten zu erhalten.

- Die installierten Pakete sollten alle auf den neuesten Stand geupdated werden. Beim späteren Betrieb muss ein Administrator bei bekannt werdenden Sicherheitslücken sofort reagieren und entsprechende Patches installieren.

- Zum Schutz vor "Trojanern" könnten ausgehende Verbindungen gesperrt werden.

- Mit einer Mindestlänge für Benutzer-Passwörter können Benutzer dazu gezwungen werden, keine allzu simplen Passwörter zu verwenden. Somit sind die Daten einzelner Benutzer besser geschützt.

- Durch eine Sessionsverwaltung und das automatische Ausloggen der Benutzer nach einer gewissen Zeit der Inaktivität könnte das Risiko einer Verwendung eines nicht geschlossenen Browsers durch Unbefugte verkleinert werden. Wie dies bei Authentifikation mittles Zertifikaten geschehen soll, ist allerdings unklar.

Review des Systems von M. Payer, F. Schütz, M. Deluigi, D. Hottinger

Das untersuchte System ist in vielen Teilen dem unseren sehr ähnlich. Eingesetzt werden MySQL, Apache, OpenSSL und Perl (mit den Modulen DBI und DBD::mysql). Das Betriebssystem ist ein aktuelles Debian. Auf den Einsatz eines zweiten Rechners für Backups wurde aus Zeitgründen verzichtet. Der einzige wesentliche Unterschied betrifft die Benutzerauthentifikation. Während in unserem System jeder Zugriff separat authentiziert wird (aufgrund eines Zertifikats oder mit den Login-Daten des Benutzers), werden im untersuchten System nach einer einmalig erfolgreichen Authentifikation Cookies eingesetzt.

Unser Augenmerk richten wir auf dieselben Punkte wie in der Sicherheitsanalyse unseres eigenen Systems:

1. Programmierfehler

Klassische Probleme wie Buffer Overflows werden durch die Verwendung der Programmiersprache Perl mit automatischem Speicher effektiv verhindert. Der *Strict-* und der *Taint-*Modus werden ebenfalls konsequent eingesetzt. Positiv ist auch anzumerken, dass das untersuchte System durchgehend objektorientiert programmiert und damit sehr übersichtlich ist. Das untersuchte Team setzt ebenfalls auf die Verwendung der häufig verwendeten Module DBI und DBD::mysql. In Punkt 1 konnten wir deshalb keine sicherheitsrelevanten Lücken feststellen.

Keine Sicherheitslücke, aber unserer Ansicht nach eine Unschönheit stellt die Tatsache dar, dass eine Änderung der Personenangaben zuvor gültige Zertifikate plötzlich ungültig macht.

2. Angriffe auf das System

Ein `nmap -O -sV -v seclab` ergibt folgende Ausgabe, d.h. Versuche, TCP-Sequenznummern für dieses System zu erraten, sind sinnlos:

```
OS details:  Linux 2.4.20 - 2.4.22 w/grsecurity.org patch,
Linux 2.4.22-ck2 (x86) w/grsecurity.org and HZ=1000 patches
TCP Sequence Prediction:  Class=truly random
Difficulty=9999999 (Good luck!)
PID Sequence Generation:  Randomized
```

Apache gibt Versionsinformationen (auch von mod_ssl und Openssl) aus:

```
Apache/1.3.26 (Unix) Debian GNU/Linux mod_ssl/2.8.9 OpenSSL/0.9.6c
```

Aktuell (10.01.2005) ist Apache 1.3.33 Zwischen diesen Versionen gibt es diverse Patches, welche Sicherheitsrelevante Schwachstellen anpassen. Wir vermuten allerdings, dass diese Schwachstellen bei der älteren Version nachträglich gepatched wurden.

Es ist ein Paketfilter vorhanden, jedoch nicht konfiguriert. Der Einsatz des Paketfilters könnte die Sicherheit unserer Meinung nach erhöhen, etwa bei der späteren Installation zusätzlicher Software. Bei der aktuellen Softwarekonfiguration wird er nicht benötigt.

Weiter ist SSH über das Netzwerk erreichbar:

```
debug1:  Remote protocol version 2.0, remote software version
OpenSSH_3.4p1 Debian 1:3.4p1-1.woody.3
debug1:  match:  OpenSSH_3.4p1 Debian 1:3.4p1-1.woody.3 pat OpenSSH*

debug1:  Enabling compatibility mode for protocol 2.0
```

Ein versuch mit SSH-Version 1, dem älteren, verwundbaren Protokoll schlug fehl, da dies in der /etc/ssh/sshd_config korrekt deaktiviert wurde. Angriffe auf den SSH-daemon sind damit mit hoher Wahrscheinlichkeit unmöglich.

Mit Nessus (Version 2.0.12) konnten wir folgende Sicherheits-Warnungen erhalten:

- *ICMP Timestamp request:* Ein Angreifer kann den ICMP-timestamp-Service benutzen, um die Systemzeit zu erhalten. Dies ist insbesondere bei der weiter unten vorgestellten Attacke auf Session-IDs hilfreich.
- *TCP:* TCP-SYN-Packete mit gesetztem FIN Flag werden nicht verworfen. Dies könnte je nach Firewallsoftware benutzt werden, um Regeln der Firewall zu umgehen. Für das untersuchte System besteht hierdurch aber keine grosse Gefahr.
- *HTTPS:* Versionen von mod_ssl und apache. Diverse Sicherheitsbugs sind in den verwendeten Versionen entdeckt worden. Wir vermuten allerdings, dass diese in den neuen Debian-Releases beseitigt worden sind.
- *HTTPS:* Apache unterstützt in der verwendeten Konfiguration die TRACE und TRACK-Methoden. Diese werden normalerweise mit Schwachstellen in diversen Browsern ausgenutzt, um X-Site-Scripting-Attacken auf andere Systeme zu fahren. Hiermit könnte z.B. mit der unten vorgestellten Attacke die Session-ID eines Benutzers ausgelesen werden und dann genutzt werden, um von der gleichen IP eine Session zu uebernehmen.
- *HTTPS:* Mit der verwendeten Apache-Version können beliebige escape-Sequenzen in das error-Logfile eingefügt werden. Hiermit könnte ein Administrator, der das Log in einem Terminal betrachtet, erfolgreich getäuscht werden. Risiko: Klein

Nessus findet zusätzlich das folgende Sicherheits-Loch:

- *HTTPS:* In der verwendeten Version des Webservers sind Schwachstellen in den Modulen mod_alias und mod_rewrite vorhanden. Dies könnte einem Gegner erlauben, beliebigen Code auf dem Server auszufuehren. Risiko: Hoch (CAN-2003-0542[2]). Das Sicherheitsloch dürfte allerdings nicht von Bedeutung sein, da diese Module nicht verwendet werden.

Bei der Konfiguration des Servers wurde erheblichen Wert darauf gelegt, dass es keine Symlinkschwachstellen und keine Files mit den falschen Berechtigungen gibt. Zudem wurden alle Rechte soweit wie möglich eingeschränkt. Eine Analyse aller geänderten Dateien seit dem 6. November 17:02 bestätigte diese Aussage von den Entwicklern. In der .mysql_history sind sämtliche GRANT-Befehle, welche auf dem MySQL ausgefuehrt wurden, aufgelistet. Dieses File ist nur als root lesbar und somit eigentlich keine eigentliche Gefahr. Mit root-Privilgien ist es ohnehin ein leichtes, Zugriff auf die Datenbank zu erhalten.

[2] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0542

Bei der Untersuchung der Angreifbarkeit des Systems wirkt es sich nachteilig aus, dass die Benutzer-Authentifikation nicht bewährter Software wie Apache und MySQL überlassen wird. Die Authentifikations-Mechanismen wurden selber programmiert. Während wir bei der Authentifikation selbst keine Schwachstellen finden konnten, enthält die verwendete Sessionsverwaltung in der Praxis ausnutzbare Sicherheitslücken.

Das System funktioniert wie folgt. Nach erfolgreichem Login wird ein Cookie gesetzt. Dieses enthält als Wert die SHA1-Prüfsumme der Client-IP-Adresse, des Benutzernamens und der Systemzeit. Diese Daten werden in einer Datenbanktabelle gespeichert. Bei jedem neuerlichen Zugriff innerhalb der folgenden 300 Sekunden wird der Wert des Cookies mit dem in der Datenbank gespeicherten Wert verglichen. Sofern der Wert vorhanden ist, wird der Zugriff gewährt und ein neues Cookie berechnet. Obiger Mechanismus kann von einem Angreifer ausgenutzt werden, um die Authentifizierung zu umgehen und die Session eines Benutzers zu übernehmen. Dazu muss der Angreifer

- den obigen Session-Management-Mechanismus kennen
- den Benutzernamen des Benutzers kennen, dessen Session er übernehmen will
- dieselbe IP-Adresse wie der Benutzer haben

Dieses Szenario ist in Umgebungen, wo NAT verwendet wird (Schulen, Heimnetze, Firmennetze) durchaus realistisch. Da die Systemzeit des Servers einfach erhalten werden kann, muss ein Angreifer mit den notwendigen Voraussetzungen lediglich 300 der für die im Cookie verwendeten Systemzeit in Frage kommenden Werte durchprobieren. Den beschriebenen Angriff konnten wir mit folgendem Skript erfolgreich durchführen:

```
#!/bin/bash
USER="a3" TIME=‘perl -e 'print time()-300;'‘
CLIENTIP="192.168.1.1"
while true; do
    HASH=‘echo -n $CLIENTIP$TIME$USER | openssl sha1‘
    if curl -k -b "sid=$HASH" -i https://seclab/workspace/overview.pl
2>/d /null | tee /tmp/request | grep "failed" >/dev/null;
    then
        TIME=$(($TIME+1))
    else
        echo "$HASH";
        echo "$TIME";
        echo "New request:";
        cat /tmp/request;
        break;
    fi
done;
```

Gültige Zugangsdaten stehen anschliessend in /tmp/request. Damit kann ein Angreifer sich ein gültiges Zertifikat ausstellen.

Eine Untersuchung des Codes zeigt, dass man ohne Weiteres zusätzlich einen zufälligen Wert in die Berechnung der Session-ID hätte miteinbeziehn können. Würde die Session-ID beispielsweise mit

```
SHA1(ClientIP,SystemTime,UserID,16 Random Bytes)
```

berechnet, wäre dieser Angriff nicht mehr duchführbar.

Das untersuchte System ist unserer Ansicht nach gegen manipulierte Formulareingaben resistent. Etwas unschön ist es, dass man durch die Eingabe des Strings `", "` gültige Zertifikate ausstellen kann, die später jedoch vom System nicht akzeptiert werden, da derselbe String vom Authentifikationssystem als Trennsymbol interpretiert wird. Dies ist allerdings ein reines Usability-Problem und nicht sicherheitsrelevant.

Im untersuchten System wurden auch Vorkehrungen gegen Race Conditions getroffen. Bei Dateioperationen wird jeweils ein temporäres Verzeichnis, welches sechs zufällig gewählte Buchstaben enthält, erstellt. Keine Vorkehrungen wurden gegen auftretende Inkonsistenzen in der Datenbank getroffen (z.B. wenn mehrere Zugriffe für denselben Benutzer gleichzeitig auf die Datenbank geschehen). Wir sehen aber keine Möglichkeit, wie dies als Sicherheitslücke ausgenutzt werden könnte.

Die Programmierer haben teilweise vergessen, den Debug-Modus ihrer Skripte ("fatals to browser") abzuschalten, so dass ein externer Benutzer Informationen über den Aufbau der Programme erhalten könnte (Information Leaking). Dabei dürfte es sich um ein Versehen in der Eile handeln.

3. Fehlverhalten eines Anwenders

In diesem Bereich verfolgen die Programmierer des untersuchten Systems eine andere Philosophie, als wir sie verfolgt haben. Dies liegt daran, dass die Aufgabenstellung viel Freiraum lässt. Neben der oben bereits erwähnten Sicherheitslücke im Session-Handling wirft die Verwendung von Cookies Sicherheitsfragen auf. Die Programmierer haben durch die Begrenzung der Gültigkeitsdauer auf fünf Minuten und die Beschränkung auf eine IP-Adresse viel getan, um Missbrauch zu vermeiden. Allerdings sind die Cookies zumindest auf dem Mozilla-Browser teilweise noch vorhanden, wenn man sich nicht sauber ausloggt, sondern den Browser einfach schliesst und neu startet. Viele Benutzer werden dies wohl an auch von Anderen benutzten Rechnern (beispielsweise in einem Internet-Café) tun. Dabei könnte es sich aber auch um eine Schwäche des Mozilla-Browsers handeln.

Das verwendete Session-Handling hat allerdings auch positive Aspekte: Da nach Ablauf von fünf Minuten ohne Tätigkeit eine neue Authentifikation erforderlich ist, werden Personen automatisch ausgeloggt, die ihren Rechner verlassen oder die Cookiedaten hinterlassen. Dies ist gegenüber unserem eigenen System ein klarer Vorteil.

Ebenfalls ist es beim untersuchten System möglich, dass ein Benutzer beliebig viele Zertifikate haben kann. Dies hat den Vorteil, dass ein Benutzer sein Zertikiat nicht von einem Rechner auf den anderen kopieren muss, andererseits könnte es auch dazu führen, dass Benutzer ein verloren gegangenes Zertifikat einfach durch ein neues ersetzten, ohne das alte zu widerrufen. Dies birgt ein gewisses Missbrauchspotential. Andererseits verlangt das untersuchte System im Gegensatz zu unserem eine Mindestlänge beim Passwort, was für die Sicherheit der Benutzer möglicherweise vorteilhaft ist.