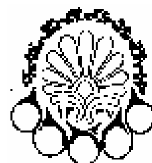
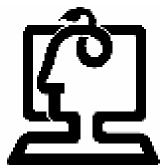
www.sazadi.com



دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)



دانشکده‌ی مهندسی کامپیوتر
و فناوری اطلاعات

پروژه‌ی کارشناسی مهندسی کامپیوتر
گرایش نرم‌افزار

عنوان پروژه:

تشخیص دستکاری غیر مجاز پرونده‌های اجرایی سیستم عامل لینوکس توسط تولید کدهای تشخیص دستکاری بازگشت‌ناپذیر

توسط:

سیامک آزادی ابد

استاد راهنما:

آقای دکتر حسین پدرام

شهریور ماه ۱۳۸۵

تقدیم به

پدر و مادر عزیزم

صمیمانه از **جناب آقای دکتر پدرام** و **جناب آقای دکتر صادقیان** که طی انجام پروژه از یاری ایشان استفاده کردم تشکر و قدردانی می‌نمایم.

از **سرکار خانم محدثه سوفالی** که زحمات بسیاری در ویراستاری نگارشی و فنی پایان‌نامه متحمل شدند نیز سپاس‌گذارم.

چکیده

امروزه و در عصر اطلاعات، محافظت از اطلاعات اهمیت بالایی دارد، و می‌دانیم تمام اطلاعات ذخیره‌شده روی سیستم‌های کامپیوتری، درنهایت روی سیستم فایل قرار می‌گیرند. ازطرفی برای اعتماد به اطلاعات موجود و نیز کارکرد خود سیستم، باید بتوانیم صحت اطلاعات و برنامه‌های سیستم را بررسی و تایید کنیم.

در این پروژه، ابزارهای بررسی صحت معرفی و مشکلات آن‌ها بررسی می‌شوند. درنهایت راه‌حل‌هایی نیز برای برخی از آن مشکلات ارایه می‌شود.

فهرست مطالب

مقدمه ۱

۱- آشنایی اولیه ۴

۱-۱- روش‌های ایجاد درب پشتی ۵

۱-۱-۱. استفاده از برنامه‌های مشتری-خدمتگزار ۵

۱-۱-۲. استفاده از Rootkitهای سنتی ۸

۱-۱-۳. استفاده از Rootkitهای سطح هسته ۱۴

۱-۲- اهمیت بررسی صحت و جامعیت اطلاعات و پرونده‌ها ۱۸

۲- بررسی‌کننده‌های صحت پرونده‌ها ۲۱

۲-۱- معرفی اولیه ۲۲

۲-۲- نحوه‌ی کار ابزارهای بررسی صحت و ساختار آن‌ها به‌صورت کلی ۲۴

۲-۳- خلاصه‌ی پیام و الگوریتم MD5 .... ۲۸

۲-۳-۱. الگوریتم MD5 .... ۲۹

۳- بررسی محدودیت‌ها و مشکلات ۳۲

۳-۱- تقلید امضا ۳۳

۳-۲- تشخیص دستکاری پس از ایجاد تغییرات ۳۶

۳-۳- عدم استقلال برنامه‌ی تشخیص دستکاری از سیستم عامل ۳۷

۳-۴- نحوه‌ی ذخیره‌ی اطلاعات در پایگاه داده ۳۹

۴- راه‌حل و ساختار پیشنهادی ۴۱

۴-۱- محدوده‌ی پروژه ۴۲

۴-۲- معرفی سیستم عامل اوبانتو ۴۳

۴-۳- لوح فشرده‌ی زنده ۴۴

۴-۴- استقلال برنامه‌ی تشخیص دستکاری از سیستم عامل ۴۶

۴-۵- مسایل مربوط به پایگاه داده ... ۴۸
۴-۶- نحوه‌ی کار برنامه‌ی تشخیص دستکاری پرونده های اجرایی ... ۵۰
۴-۶-۱. معماری سیستم ... ۵۰
۴-۶-۲. نحوه‌ی کار برنامه ... ۵۱
۴-۶-۳. نحوه‌ی استفاده از برنامه ... ۵۴
مراجع ... ۵۵
پیوست‌ها ... ۵۷
پیوست الف: RFC 1321 (الگوریتم MD5) ... ۵۸

فهرست شکل‌ها

۱-۱: نحوه‌ی اتصال به قسمت خدمتگزار VNC ۶
۱-۲: دسترسی به صفحه‌ی کاری کاربر مدیر سیستم ۷
۱-۳: مقایسه‌ی درب‌های پشتی سطح برنامه‌ی کاربردی با Rootkitهای سنتی ۹
۱-۴: محتوای پرونده‌ی original.txt ۱۱
۱-۵: محتوای پرونده‌ی modified.txt ۱۲
۱-۶: مقایسه‌ی original.txt با modified.txt توسط دستور diff ۱۲
۱-۷: نمایش آخرین زمان تغییر پرونده با استفاده از دستور ls ۱۳
۱-۸: معماری کلی یونیکس ۱۵
۱-۹: هسته‌ی یونیکس ۱۶
۱-۱۰: مقایسه‌ی Rootkitهای سنتی با Rootkitهای سطح هسته ۱۷
۲-۱: نحوه‌ی ارتباط اجزای سیستم بررسی صحت سیستم فایل ۲۷
۴-۱: اجزای برنامه‌ی تشخیص دستکاری پرونده‌های اجرایی و ارتباط‌های آن‌ها ۵۰
۴-۲: صفحه‌ی اصلی اجرای برنامه‌ی تشخیص دستکاری ۵۲
۴-۳: فلوچارت روند اجرای برنامه‌ی تشخیص دستکاری ۵۳

فهرست جدول‌ها

۳-۱: تعداد دفعات تصادم امضا برای الگوریتم‌های مختلف ۳۵

مقدمه

به‌طور کلی تقسیم‌بندی‌های مختلفی از دیدگاه‌های متفاوت برای مشکلات امنیتی سیستم‌های کامپیوتری درنظر گرفته شده است. یکی از آن تقسیم‌بندی‌ها، دسته‌بندی انجام‌شده توسط پروفسور تننبام [۱] است که مشکلات امنیتی را به چهار رده‌ی نزدیک به هم تقسیم می‌کند: سری ماندن اطلاعات یا محرمانگی[1]، احراز هویت کاربران[2]، غیرقابل‌انکار بودن پیام‌ها[3] و نظارت بر صحت اطلاعات[4].

"سری ماندن اطلاعات" متضمن انجام عملیاتی است که اطلاعات را از دسترس کاربران غیرمجاز دور نگه می‌دارد.

"احراز هویت" عبارت است از تأیید هویت طرف مقابل ارتباط، قبل از آن‌که اطلاعات حساس در اختیار او قرار گیرد.

"غیرقابل‌انکار بودن پیام‌ها" با امضای دیجیتالی سروکار دارد و به اطلاعات و مستندات دیجیتالی، هویت حقوقی اعطا می‌کند.

حال چگونه می‌توان مطمئن شد پیامی که شما دریافت کرده‌اید دقیقا همان پیامی است که در اصل فرستاده شده و کسی آن را دستکاری و تحریف نکرده است؟ یا برنامه‌ای را که شما همیشه روی یک سیستم کامپیوتری از آن استفاده می‌کنید، کسی تغییر نداده تا در زمان اجرا اعمال دیگری را در

[1] - Secrecy (Confidentiality)
[2] - Authentication
[3] - Non Repudiation
[4] - Integrity Control

کنار کار شما انجام دهد؟ این مسایل و موارد مشابه آن موضوع بحث در رده‌ی "نظارت بر صحت اطلاعات" هستند.

در این پایان‌نامه، به موضوع "نظارت بر صحت اطلاعات" برنامه‌های اجرایی سیستم عامل لینوکس پرداخته شده است. ابتدا درمورد مسایل کلی و فناوری موجود کنونی بحث شده، سپس برخی مشکلات آن بررسی شده اند و سعی شده این مشکلات حل شوند.

فصل اول:

آشنایی اولیه

۱-۱- روش‌های ایجاد درب پشتی[1]

غالبا زمانی که یک نفوذگر کامپیوتری با سوءاستفاده از یک مشکل امنیتی به یک سیستم نفوذ می‌کند، می‌خواهد که آن سیستم را برای همیشه برای خود حفظ کند. لذا به‌دنبال راهی می‌گردد تا بتواند دسترسی خود را به آن سیستم همیشگی کند. برای رسیدن به این هدف، مهاجمان از تکنیک‌های مختلفی استفاده می‌کنند که به‌طور کلی به آن‌ها ایجاد "درب پشتی" می‌گویند.

ایجاد "درب پشتی"، همان‌طور که از نامش پیداست، به مهاجم اجازه می‌دهد که راه‌های ورودی دیگری جز راهی که اولین بار از آن به سیستم نفوذ کرده است، ایجاد کند. هنگامی‌که مهاجمان یک درب پشتی را روی سیستمی ایجاد می‌کنند، می‌توانند بدون استفاده از کلمات عبور و ساختارهای تصدیق هویت[2] به سیستم دسترسی پیدا کنند. [۲]

در زیر به مثال‌هایی از روش‌های ایجاد درب پشتی اشاره می‌شود:

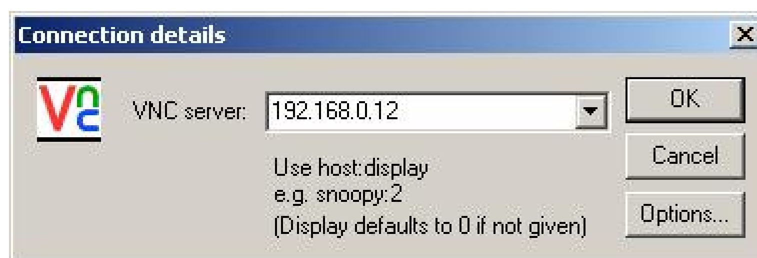
۱-۱-۱. استفاده از برنامه‌های مشتری-خدمت‌گزار[3]

یکی از روش‌هایی که مهاجمان برای ایجاد درب پشتی از آن استفاده می‌کنند، استفاده از برنامه‌های درب پشتی است که غالبا از دو بخش مشتری یا سرویس‌گیرنده، و خدمت‌گزار یا سرویس‌دهنده تشکیل شده‌اند. مهاجم پس از نفوذ به سیستم، بخش خدمت‌گزار آن را روی سیستم موردنظر نصب و اجرا می‌کند. غالبا این برنامه به‌صورت سرویس به اجرا در می‌آید و روی پورت

1 - Back Door
2 - Authentication
3 - Client-Server

خاصی روی شبکه گوش فرا می‌دهد تا اگر درخواستی از سمت سرویس‌گیرنده برسد، به آن پاسخ دهد.

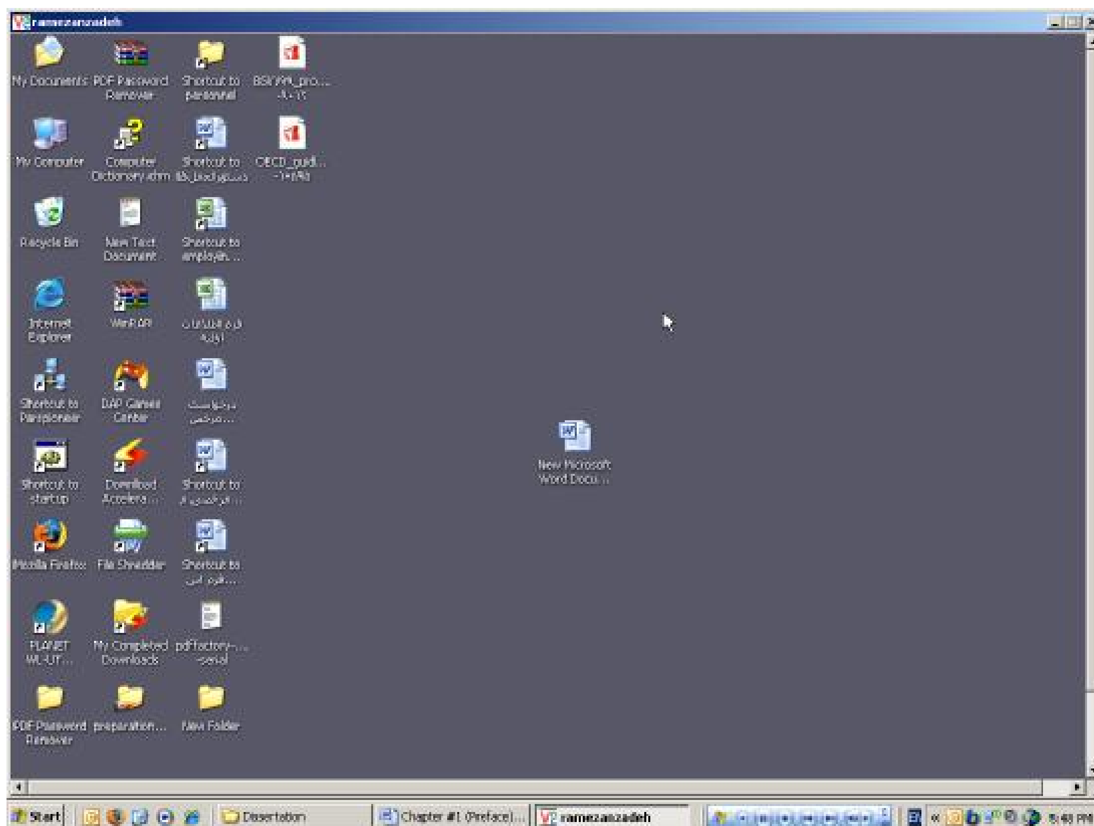
یکی از ابزارهای معروفی که هم مدیران شبکه و هم مهاجمان کامپیوتری از آن استفاده می‌کنند، ابزار VNC است. این ابزار می‌تواند به حالت خدمت‌گزار روی سیستم نصب شود و هرگاه برنامه‌ی مشتری به آن وصل شود، امکان کامل استفاده از سیستم را به شخص می‌دهد؛ به این صورت که به‌صورت راه‌دور[1] می‌توان به صفحه‌ی نمایش سیستم دسترسی یافت. مدیران سیستم از این ابزار جهت مدیریت راه دور، ولی مهاجمان نیز از آن به عنوان درب پشتی استفاده می‌کنند که در صورت عدم آگاهی مدیر سیستم، برای مهاجم بسیار کارآمد و قوی خواهد بود. نسخه‌های مختلفی از این ابزار برای سیستم‌های عامل مختلف مانند ویندوز و لینوکس وجود دارند. در شکل ۱-۱ نمایی از اجرای قسمت مشتری این ابزار را مشاهده می‌کنید.



شکل ۱-۱: نحوه‌ی اتصال به قسمت خدمت‌گزار VNC

[1] - Remote

پس از اجرا و وصل شدن به برنامه‌ی خدمت‌گزار، همانطور که در شکل ۱-۲ نمایش داده شده است، نفوذگر سیستم به همه‌ی منابع و درحقیقت به کل سیستم از راه دور دسترسی دارد.



شکل ۱-۲: دسترسی به صفحه‌ی کاری کاربر مدیر سیستم

به‌طور کلی همه‌ی این‌گونه ابزارها، چه ابزارهای آماده و شناخته‌شده و چه ابزارهایی که در گروه‌های زیرزمینی ایجاد شده‌اند، یک ویژگی مشترک و اساسی دارند: این‌که باید پرونده‌ها یا برنامه‌هایی را روی سیستم مورد نفوذ ایجاد کنند.

از راه‌های تشخیص وجود این ابزارها، استفاده از ابزارهای ضدویروس و در بعضی مواقع برنامه‌های ضدجاسوسی[1] است. این ابزارها می‌توانند برنامه‌های معروفی را که به‌عنوان درب پشتی شناخته شده‌اند، تشخیص و به شما گزارش دهند. ولی اگر از یک برنامه‌ی ناشناخته استفاده شود، راه‌حل چیست؟ در این‌گونه موارد باید روش یا ابزاری برای تشخیص وجود برنامه‌های جدید ناخواسته وجود داشته باشد تا بتوان تغییرات ایجادشده در سیستم فایل را تشخیص داد. [۲]

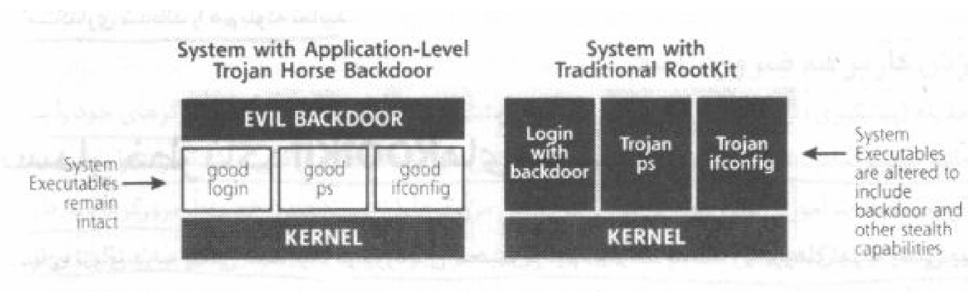
البته روش دیگری نیز برای یافتن آثاری از این برنامه‌ها وجود دارد و آن پویش پورت سیستمی است که مورد نفوذ واقع شده است. اگر برنامه‌ی سرویس‌دهنده پورتی را روی سیستم باز کند، با یک پویش پورت ساده قابل تشخیص خواهد بود. ولی متأسفانه همیشه کار به این سادگی نیست؛ چراکه بعضی از این نرم‌افزارها با شرایط خاصی کار می‌کنند. برای مثال، به مدت چند دقیقه در ساعات خاصی از طول شب فعال می‌شوند. بهترین و مطمئن‌ترین راه تشخیص، بررسی سیستم فایل برای تشخیص دستکاری‌های غیرمجاز است.

۱-۱-۲. استفاده از Rootkitهای سنتی

برنامه‌های درب پشتی که در بخش قبل معرفی شدند، توسط مهاجمان سیستم‌های کامپیوتری زیاد مورداستفاده قرار می‌گیرند، اما اغلب به‌راحتی قابل‌شناسایی هستند و اگر مدیر سیستم به وضعیت سیستم خود مشکوک شود به آسانی (در اکثر موارد) می‌تواند وجود این برنامه‌ها را تشخیص دهد. [۲]

[1] - **Anti Spy**

Rootkitهای سنتی با تغییر یا جایگزینی اجزای سیستمی موجود در سیستم فایل، همان‌طور که در شکل ۱-۳ نشان داده شده است، می‌توانند قدرت بیشتری از خود نشان دهند. [۲]



شکل ۱-۳: مقایسه‌ی درب‌های پشتی سطح برنامه‌ی کاربردی با Rootkitهای سنتی

Rootkitها علاوه بر این‌که به‌عنوان یک برنامه‌ی کاربردی درب پشتی اجرا می‌شوند، جایگزین برنامه‌های اجرایی سیستم عامل می‌شوند. به‌این‌ترتیب هیچ برنامه‌ی اضافی‌ای روی سیستم اضافه نمی‌شود، بلکه این برنامه‌های اجرایی و پرونده‌های خود سیستم است که تغییر کرده‌اند. [۲]

اولین Rootkit در سال ۱۹۹۰ میلادی به‌صورت عمومی در مجامع کامپیوتری شناخته شد و ازآن‌پس بسیار مورد علاقه‌ی نفوذگران و انجمن‌های زیرزمینی قرار گرفت.

این Rootkitها برای اغلب سیستم‌های عامل مختلف وجود دارند، ولی از دیرباز روی سیستم‌های عامل یونیکس، مانند لینوکس، سولاریس و... (شاید به‌علت متن- باز[1] بودن برنامه‌های آن‌ها) تمرکز دارند. [۲]

[1] - Open Source

یکی از این نوع Rootkitها، برنامه‌ی تغییرداده‌شده‌ی sh یا bash است. در سیستم‌های لینوکس، این برنامه‌ها وظیفه‌ی دراختیار قرار دادن پوسته‌ی [۱] سیستم عامل را به کاربر به‌عهده دارند و دستوراتی را که کاربر می‌دهد برای اجرا به سطوح پایین‌تر می‌دهند. این برنامه‌ها همچنین با "واسط سطح کاربری و سطح هسته[۲]" در ارتباط هستند و دستوراتی را که باید در سطح هسته اجرا شوند به این واسط تحویل می‌دهند. [۲]

یکی از ویژگی‌های sh یا bash این است که بسته به سطح دسترسی کاربری که هم‌اکنون این برنامه‌ها را اجرا کرده، اجازه‌ی اجرای انواع دستورات و نیز اجازه‌ی دسترسی به انواع منابع را (البته از طریق واسطه‌های مختلف) به کاربر اعطا می‌کند. برای مثال اگر برنامه‌ی bash با سطح دسترسی مدیر سیستم[۳] به اجرا در آید، اجازه‌ی دسترسی به تمامی پرونده‌های سیستم را به کاربر خواهد داد. ولی اگر با سطح دسترسی یک کاربر عادی[۴] اجرا شود، اجازه‌ی دسترسی به برخی پرونده‌ها مانند اطلاعات کلمه عبور کاربران را نخواهد داد.

در این مثال، مهاجم برای Rootkit کردن سیستم، برنامه‌ی اجرایی bash را طوری تغییر می‌دهد که به کاربران معمولی یا یک کاربر معمولی خاص، سطح دسترسی مدیر سیستم داده شود. مثال‌های زیادی در این زمینه وجود دارند و ساخت و استفاده از این‌گونه ابزارها امروزه در حال گسترش است.

---

[1] - Shell
[2] - Kernel and User Level Interface
[3] - Root یا System Administrator
[4] - User

برای تشخیص این‌که آیا سیستمی توسط Rootkitهای سنتی تغییر یافته است، تنها راه موجود بررسی صحت پرونده‌های سیستم است. باید روش یا ابزاری برای بررسی صحت پرونده‌های سیستم وجود داشته باشد تا بتوان تغییرات ایجادشده را تشخیص داد؛ به‌این‌صورت که با مقایسه‌ی پرونده‌ها به روش‌های مختلف مانند مقایسه با پرونده‌های اصلی پشتیبان گیری شده، و یا با بررسی برخی ویژگی‌های پرونده مانند آخرین زمان تغییر پرونده (که سیستم عامل این زمان را به‌روز نگه می‌دارد) می‌توان به این مسئله پی برد. در شکل ۱-۶ مقایسه‌ی یک پرونده با پرونده‌ی اصلی نشان داده می‌شود. پرونده‌ی اصلی پرونده‌ای متنی با محتوای سه خط متنی بوده است. همان‌طور که مشاهده می‌شود، مهاجم این پرونده را تغییر داده و این مسئله با مقایسه توسط دستور diff در سیستم عامل لینوکس به‌راحتی قابل‌تشخیص است.

```
root@Beh: /tmp
File Edit View Terminal Tabs Help
This is a simple test.
This file will be modified by an attacker.
We use "diff" command to compare 2 files.
~
"original.txt" 3L, 108C                                1,1          All
```

شکل ۱-۴: محتوای پرونده ی original.txt

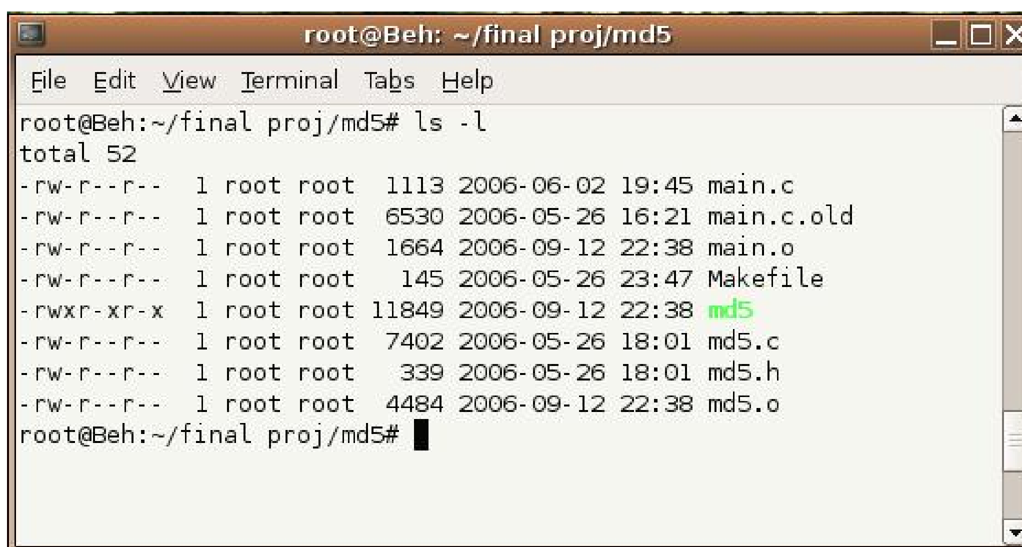
```
root@Beh: /tmp
File  Edit  View  Terminal  Tabs  Help
This is a simple test.
This is the modified version.
We use "diff" command to compare 2 files.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"modified.txt" 3L, 95C                                    1,1          All
```

شکل ۱-۵: محتوای پرونده ی modified.txt

```
root@Beh: /tmp
File  Edit  View  Terminal  Tabs  Help
root@Beh:/tmp# diff original.txt modified.txt
2c2
< This file will be modified by an attacker.
---
> This is the modified version.
root@Beh:/tmp#
```

شکل ۱-۶: مقایسه ی original.txt با modified.txt توسط دستور diff

با استفاده از دستور ls در سیستم عامل لینوکس، می‌توان آخرین زمان تغییر پرونده را مشاهده کرد. نحوه‌ی استفاده از این دستور برای تشخیص دستکاری در شکل ۱-۷ نشان داده شده است. برای مثال آخرین زمان تغییر پرونده ی main.c در شکل، ساعت ۱۹:۴۵ در تاریخ ۲۰۰۶/۶/۲ است.



شکل ۱-۷: نمایش آخرین زمان تغییر پرونده با استفاده از دستور ls

۱-۱-۳. استفاده از Rootkitهای سطح هسته

Rootkitهای سنتی که در بخش قبلی توضیح داده شدند، توسط ابزارهای بررسی‌کننده‌ی صحت[1] قابل تشخیص هستند. این ابزارها بر پایه‌ی فناوری اثر انگشت دیجیتالی[1] و ساخت خلاصه‌ی-

[1] - **Integrity Checker**

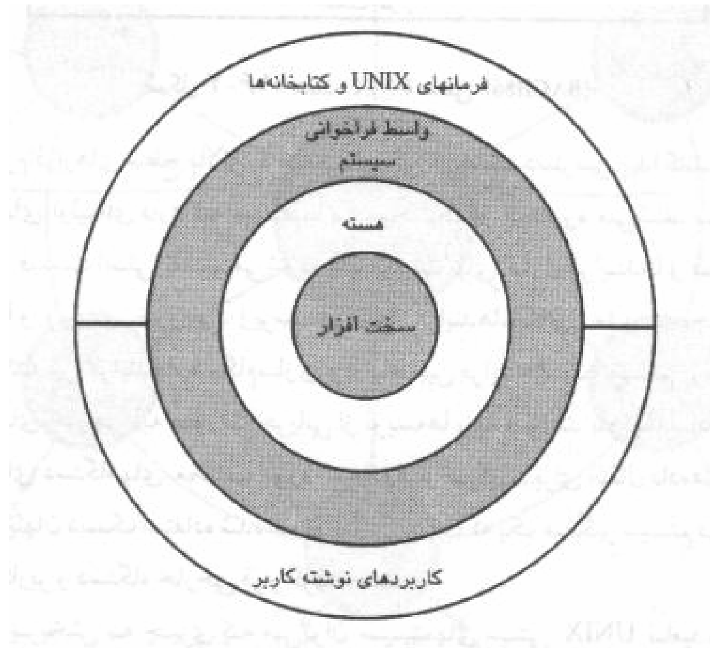
پیغام[2] کار می‌کنند، روی سیستم مورد نظر نصب شده و می‌توانند در تشخیص تغییرات ایجاد شده در پرونده‌ها و برنامه‌های جدید به مدیر سیستم کمک کنند.[۲] (برای کسب اطلاعات بیشتر در رابطه با نحوه‌ی عملکرد این ابزارها می‌توانید به بخش ۲-۲ مراجعه کنید.)

نوع دیگری از Rootkitها وجود دارند که به Rootkitهای سطح هسته[3] معروفند و تقریبا قوی‌ترین نوع برنامه‌های درب پشتی را تشکیل می‌دهند. مرحله‌ی تکامل Rootkitها، پا را فراتر از جایگزینی برنامه‌های سیستمی گذاشته است و امروزه Rootkitها در سطح هسته سیستم عامل عمل می‌کنند. [۲]

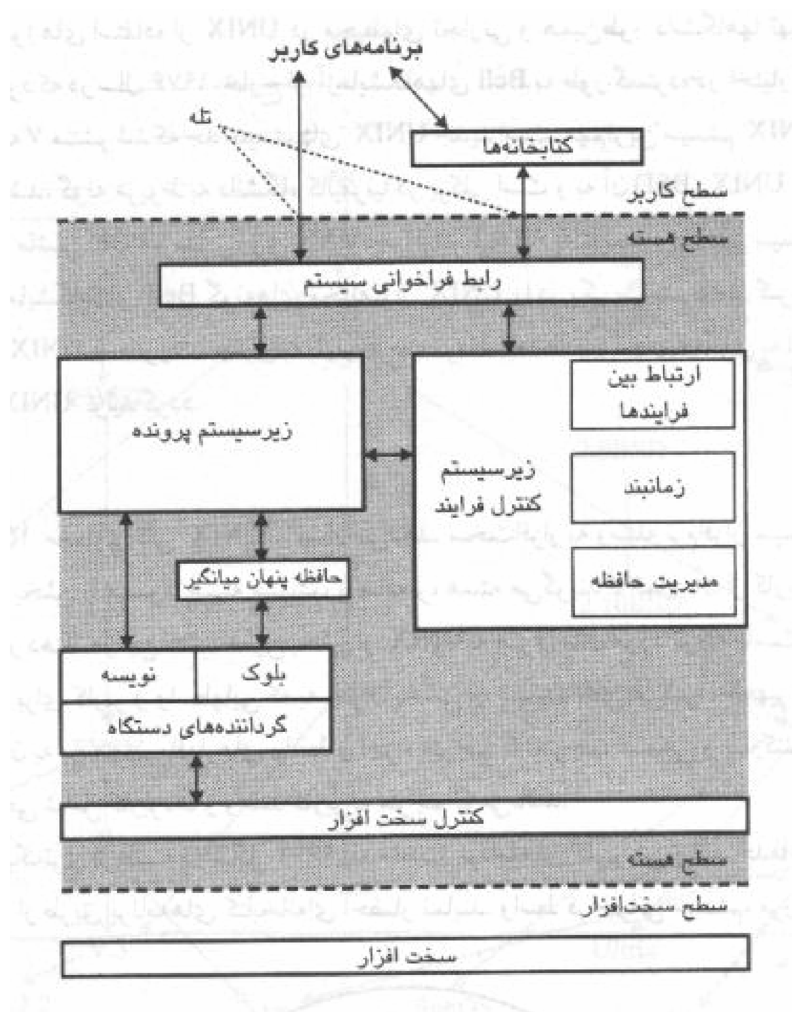
در بسیاری از سیستم‌های عامل (مانند ویندوز و سیستم‌های عامل مختلف یونیکس)، هسته یک بخش زیربنایی از سیستم عامل است که ابزارهای دسترسی به شبکه، پردازش‌ها، حافظه‌ی اصلی، حافظه‌ی سخت و غیره را کنترل می‌کند. همه‌ی منابع سیستمی توسط هسته‌ی سیستم عامل کنترل و هماهنگ می‌شوند و تمام دستوراتی که به این منابع نیاز دارند، باید از کانال هسته برای اجرای فرامین استفاده کنند. (به شکل ۱-۸ و ۱-۹ رجوع شود) [۳]

---

1 -Digital Fingerprint
2 -Message Digest
3 -Kernel Level Rootkit



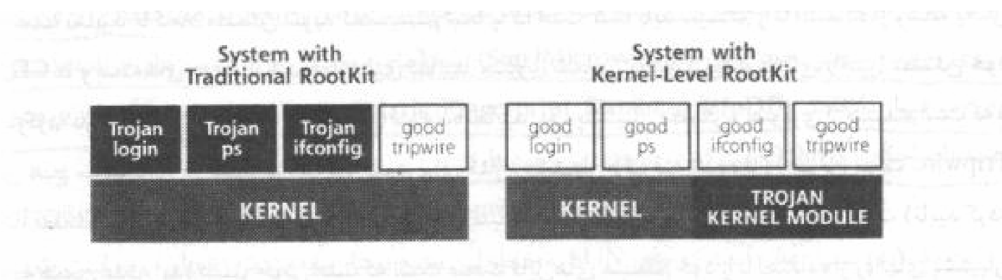
شکل ۱-۸: معماری کلی یونیکس



شکل ۱-۹: هسته‌ی یونیکس

یکی از وظایف سیستم عامل، کنترل دسترسی به پرونده‌های سیستم است. سیستم عامل برای محافظت از پرونده‌ها و کنترل دسترسی به آن‌ها، از راهکارهای حفاظتی لازم استفاده می‌کند و هیچ کاربری به‌طور مستقیم امکان و اجازه‌ی دسترسی به پرونده ای را بدون اجازه‌ی سیستم عامل ندارد. [۳]

Rootkitهای سطح هسته برای این‌که امکانات درب پشتی را فراهم کنند، هسته‌ی سیستم عامل را نیز دستکاری می‌کنند و به‌این‌ترتیب عناصر تست‌کننده یا مؤثر در تست صحت پرونده‌ها را نیز تغییر می‌دهند. (به شکل ۱-۱۰ مراجعه شود) [۲]



شکل ۱-۱۰: مقایسه‌ی Rootkitهای سنتی با Rootkitهای سطح هسته

یک مهاجم با ایجاد تغییرات اساسی در خود هسته، می‌تواند سیستم را در سطح بسیار بالایی کنترل کرده و قدرت زیادی را به‌دست آورد. همان‌طور که پیش‌تر گفته شد، سیستم عامل وظیفه‌ی کنترل دسترسی پرونده‌ها را به‌عهده دارد و تنها راه دسترسی یک کاربر یا برنامه به پرونده‌ها، سیستم مدیریت پرونده در سیستم عامل است. درنتیجه حتی برنامه‌ها و روش‌های تشخیص دستکاری نیز به‌ناچار باید از سیستم عامل درخواست کنند تا لیست پرونده‌ها و دیگر مشخصات را در اختیار آن‌ها قرار دهد. زمانی که هسته‌ی خود سیستم عامل دستکاری شده باشد، دیگر نمی‌شود به نتیجه‌ی هیچ برنامه‌ای که روی سیستم به اجرا در می‌آید اعتماد کرد، چراکه برنامه‌ی بررسی صحت پرونده‌ها، حتی برای لیست‌کردن پرونده‌های موجود، مجبور است که درخواست خود را به سیستم عامل ارایه دهد و

اگر هسته‌ی سیستم عامل و واسط‌های حالت کاربری و حالت هسته، برخی پرونده‌ها را برای مثال اصلا نشان ندهند، دیگر راهی برای تشخیص نخواهد بود. در مرجع [۲]، از نوشته‌های آقای اسکادیس پس از عنوان چند روش تقریبی برای یافتن شواهدی از وجود Rootkitهای سطح هسته، این‌گونه برداشت می‌شود که: ″درصورتی‌که این برنامه یا روش درب پشتی به‌صورت دقیق و حرفه‌ای انجام شده و از برنامه‌ای شناخته‌شده و معروف نیز استفاده نشود، در نهایت راهی برای تشخیص وجود این نوع درب پشتی وجود ندارد و علت این است که هر روش تشخیصی به‌ناچار نیاز به کمک سیستم عامل خواهد داشت، درحالی‌که خود سیستم عامل (و هسته‌ی آن) تغییر یافته است.″

در پروژه‌ی انجام شده با عنوان این پایان‌نامه، روشی برای تشخیص این گونه تغییرات در سیستم ارایه شده است. برای کسب اطلاعات بیشتر می‌توانید به بخش‌های ۳-۳، ۴-۴ و ۴-۶ مراجعه کنید.

۱-۲- اهمیت بررسی صحت و جامعیت اطلاعات و پرونده‌ها

با بحث صورت‌گرفته در بخش ۱-۱، تا کنون به اهمیت و حساسیت و دشواری بررسی صحت پرونده‌ها پی‌برده‌اید، ولی این تمام مسئله نبوده است. بررسی صحت پرونده‌ها تنها برای تشخیص درب‌های پشتی به کار نمی‌رود؛ بلکه اطمینان از صحت اطلاعات و داده‌ها یکی از مواردی است که در مباحث امنیت اطلاعات بسیار مورد تاکید قرار می‌گیرد. می‌دانیم که سیستم فایل شامل داده‌های کاربران، برنامه‌های اجرایی، پرونده‌های پیکربندی، اطلاعات مجوزهای دسترسی و غیره است.

درب‌های پشتی بر برنامه‌های اجرایی، پرونده‌های پیکربندی و اطلاعات مجوزهای دسترسی تاثیر می‌گذارند، اما امنیت اطلاعات و داده‌های کاربران نیز اهمیت بالایی دارد. فرض کنید که پرونده‌های اطلاعاتی یک سازمان (مثلا شامل قراردادها، پرونده‌های پرسنلی و...) به‌صورت فایل‌های اطلاعاتی روی کامپیوتر قرار دارند. اگر نفوذگری بتواند با دسترسی سطح بالا به سیستم وارد شود، می‌تواند صحت این اطلاعات را نیز به خطر اندازد و اگر به هدف خود برسد، نیازی به ایجاد درب پشتی برای آینده نخواهد بود.

همان‌طور که قبلا نیز به آن اشاره شد، بحث بررسی صحت و جامعیت اطلاعات در بسیاری از دسته‌بندی‌های امنیت، جایگاه خاصی دارد. برای مثال می‌توان به استاندارد BS 7799 اشاره نمود. این استاندارد بریتانیایی بوده و پایه استاندارد مشابه ISO 17799 است که به عنوان استاندارد مدیریت امنیت اطلاعات[۱] معرفی شده است. در این استاندارد می‌خوانیم: "امنیت اطلاعات به چند دسته زیر تقسیم‌بندی می‌شود:

I. محرمانگی[۲]: اطمینان از اینکه تنها افراد مجاز به اطلاعات دسترسی داشته باشند.

II. صحت (جامعیت)[۳]: محافظت از تمامیت اطلاعات و پردازش‌ها.

III. دسترسی[۴]: اطمینان از این‌که کاربران مجاز در زمان نیاز، به اطلاعات و دارایی‌های مورد نیاز دسترسی داشته باشند." [۴]

[1] - ISMS (Information Security Management System)
[2] -Confidentiality
[3] -Integrity
[4] -Availability

با توجه به توضیحات و مثال‌های بالا، حال می‌توانیم به اهمیت بررسی صحت و جامعیت اطلاعات پی بریم. در فصول بعدی به مفاهیم اصلی بررسی‌کننده‌های صحت پرونده‌ها خواهیم پرداخت.

فصل دوم:

بررسی‌کننده‌های صحت پرونده‌ها

۲-۱- معرفی اولیه

اغلب سیستم‌های کامپیوتری امروزی از منابع ذخیره‌سازی اطلاعات به‌منظور نگهداری بلندمدت داده‌ها و اطلاعات استفاده می‌کنند. به‌همین‌دلیل سیستم فایل همواره از مهمترین اهداف حملات کامپیوتری بوده است. در چنین وضعیتی، مدیران سیستم به یک سری مکانیزم نظارتی[1] نیاز دارند تا بتوانند همواره صحت و جامعیت سیستم فایل را بررسی کنند. [۵]

ازجمله ابزارهایی که به‌منظور نظارت بر صحت پرونده‌ها وجود دارند و امروزه بسیار مورداستفاده قرار می‌گیرند، بررسی‌کننده‌های صحت[2] پرونده هستند. این ابزارها اطلاعاتی از سیستم فایل را به ازای هر پرونده در پایگاه داده ذخیره می‌کنند و درصورت نیاز به بررسی صحت، با تولید دوباره‌ی آن اطلاعات و مقایسه‌ی آن‌ها (که در ساده‌ترین حالت، می‌توان یک کپی از فایل اصلی در پایگاه داده را با فایل موردبررسی مقایسه کرد)، به تغییرات اعمال شده پی ببرند. این برنامه‌ها لیستی از پرونده‌های موجود را نیز در پایگاه داده‌ی خود نگهداری می‌کنند، بنابراین می‌توانند پرونده‌های حذف[3] شده از سیستم و یا پرونده‌های جدید اضافه‌شده به سیستم را تشخیص دهند. [۶]

آن‌چه مشخص است، ابزارهای بررسی صحت به یک امضای[4] واحد[5] از پرونده‌ی موردبررسی نیاز دارند تا با مقایسه‌ی امضای قبلی و کنونی بتوانند تغییرات و دستکاری‌ها را تشخیص دهند. همان‌طور که گفته شد، یکی از این امضاها می‌تواند ذخیره‌ی یک کپی از خود پرونده باشد. این روش

---

[1] - Monitoring
[2] - Integrity Checker
[3] - Delete
[4] - Signature
[5] - Unique

از جهاتی مفید بوده و از جهاتی کارآمدی پایینی دارد. یکی از نکات مفید این روش آن است که با اطمینان مطلق می‌توان گفت امضای تولید شده واحد است. از دیگر مزایای این روش امکان بازیابی سیستم است، چراکه درصورت تشخیص دستکاری، می‌توان پرونده‌های اصلی را بازیابی کرد. اما این روش مشکلاتی نیز دربر دارد. فرض کنید ابزار موردنظر درحال تولید امضا برای کل سیستم فایل یک سیستم کامپیوتری باشد. در این روش درعمل باید یک پشتیبان[1] از کل سیستم تهیه شود. می‌بینیم که این روش فضای زیادی نیاز دارد. از طرف دیگر در زمان مقایسه نیز باید تمام پرونده مقایسه شود. این به آن معنی است که تمامی سیستم فایل باید بیت به بیت مقایسه شود و از لحاظ زمان اجرا نیز زمان قابل توجهی نیاز خواهد بود.

با توجه به آن‌چه گذشت، یک روش کارآمد برای ذخیره‌ی اطلاعات، محاسبه و تولید یک امضا با طول ثابت از محتوای پرونده موردبررسی است. اما این امضا باید به‌گونه‌ای تولید شود که وابسته به محتوای پرونده باشد (یعنی در صورت تغییر محتوا، امضا نیز تغییر کند) و از طرفی یافتن دو پرونده‌ی متفاوت با یک امضا، به اندازه‌ی کافی مشکل باشد. در آن صورت می‌توان امضای پرونده را به‌جای خود آن ذخیره کرد که نیاز به فضای کمتر دارد و مقایسه‌ی دو امضا نیز زمان بسیار کم‌تری خواهد گرفت. برای تولید امضای یک پرونده یا یک پیغام، از توابع تولید امضا یا درهم ساز[2] استفاده می‌شود. محدودیتی که توسط این توابع نسبت به روش قبلی ایجاد می‌شود این است که امضای تولیدشده بازگشت‌ناپذیر است و لذا درصورت تشخیص دستکاری، عمل بازیابی با این امضا

[1] - **Backup**
[2] - **Hash Functions**

امکان‌پذیر نخواهد بود. به امضای یک پرونده یا پیام، اثر انگشت[1] پرونده یا کد تشخیص دستکاری[2] نیز گفته می‌شود. [۵]

۲-۲- نحوه‌ی کار ابزارهای بررسی صحت و ساختار آن‌ها به‌صورت کلی

به‌طور کلی برنامه‌ها و ابزارهای بررسی صحت، شامل چند بخش اصلی هستند که در زیر به آنها اشاره شده است:

- پایگاه داده‌ی اطلاعات پرونده‌ها
- برنامه‌ی تولید کدهای تشخیص دستکاری مربوط به پرونده‌ها
- برنامه‌ی مقایسه‌ی وضعیت کنونی با اطلاعات پایگاه داده و تولید گزارش

نحوه‌ی کار ابزارهای بررسی صحت به این ترتیب است که در ابتدا (غالبا پس از نصب و اولین راه‌اندازی سیستم و قبل از ایجاد هرگونه دسترسی) یک سری امضا از پرونده‌های موردنیاز توسط برنامه‌ی تولید کدهای تشخیص دستکاری ایجاد شده و این کدها در پایگاه داده ذخیره می‌شوند. پس از آن مدیر سیستم به‌صورت دوره‌ای دوباره کدهای تشخیص دستکاری را تولید و سپس با کدهای قبلی که در پایگاه داده موجود هستند مقایسه می‌کند و در نهایت گزارشی از وضعیت سیستم آماده می‌شود. در این گزارش سه بخش مجزا وجود دارد:

---
[1] - Fingerprint
[2] - Manipulation Detection Code (MDC)

۱. لیست پرونده‌هایی که وجود نداشته‌اند و به تازگی ایجاد شده‌اند

۲. لیست پرونده‌هایی که در حال حاضر حذف شده‌اند

۳. لیست پرونده‌هایی که قبلا وجود داشته‌اند و اکنون نیز وجود دارند، ولی دستخوش تغییر شده‌اند

روند تشخیص در بخش اول و دوم از این ویژگی استفاده می‌کند که هر پرونده به‌علاوه‌ی مسیر دسترسی به آن از شاخه اصلی، دارای یک نام (به‌صورت رشته‌ی کاراکتری) واحد است،که این نام همان آدرس مطلق[1] آن پرونده از مسیر اصلی است. به عنوان مثال در سیستم‌عامل لینوکس اگر پرونده‌ای با نام book.pdf در مسیر خانه[2]ی کاربر root باشد، آدرس مطلق آن به‌صورت "/root/book.pdf" خواهد بود و این رشته‌ی کاراکتری بر روی سیستم فایل برای پرونده موردنظر واحد است؛ چرا که امکان وجود دو پرونده با نام book.pdf در مسیر "/root/" وجود ندارد و اگر دو پرونده در مسیرهای متفاوتی با نام یکسان وجود داشته باشند، آدرس مطلق آنها برای دسترسی به هریک متفاوت خواهد بود.

به همین دلایل است که برای جلوگیری از اشتباهات ناشی از یکسان بودن اسامی، برنامه‌ی تشخیص دستکاری نام همه‌ی پرونده‌ها را همراه با مسیر دسترسی آن‌ها در پایگاه داده‌ی خود ذخیره

[1] - **Absolute Address**
[2] - **Home Directory**

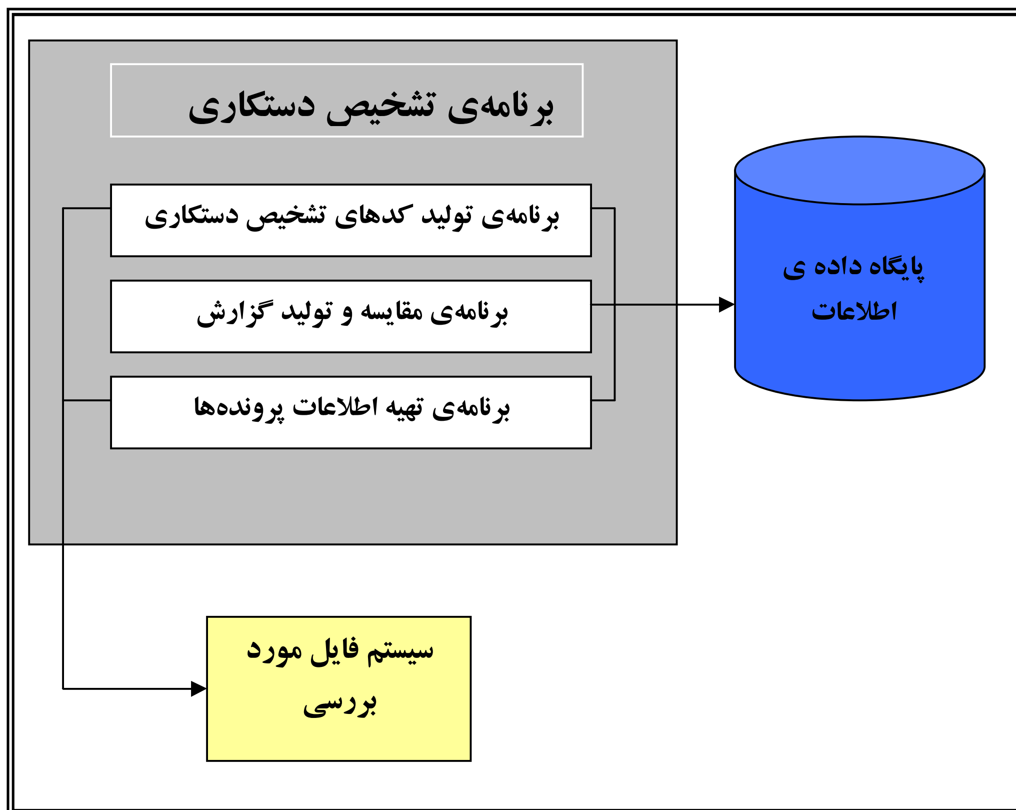
می‌کند. حال در بررسی مجدد، پس از تهیه‌ی لیست پرونده‌ها، چنان‌چه نامی وجود نداشت یا اضافه شده بود در گزارش تهیه‌شده عنوان می‌شود.

نحوه‌ی انجام کار در بخش سوم کمی با موارد قبلی متفاوت است. گفته شد که برنامه‌ی تشخیص دستکاری پس از نصب و راه‌اندازی سیستم یک بار اجرا می‌شود تا برای نخستین بار اطلاعات پرونده‌ها را در پایگاه داده وارد کند. این اطلاعات می‌توانند شامل نام پرونده، مسیر دسترسی به آن از ریشه، و کد تشخیص دستکاری تولیدشده باشند. در اجرای مجدد برنامه‌ی تشخیص دستکاری، به‌ازای هر پرونده، دوباره کد تشخیص دستکاری تولید می‌شود و می‌توان این کد را با کد قبلی که در پایگاه داده وجود دارد مقایسه کرد. از آن‌جا که کد ایجادشده به‌ازای هر پرونده یکتاست، درصورت وجود هرگونه تغییر در پرونده، کد ایجادشده متفاوت خواهد بود. به‌این‌ترتیب می‌توان دستکاری غیرمجاز پرونده‌ها را تشخیص داد و آن را گزارش کرد.

پس از گزارش‌گیری، درصورت نیاز می‌توان پایگاه داده را دوباره به‌روز کرد. یکی از دلایل انجام این کار می‌تواند این باشد که تغییر، حذف و ایجاد پرونده‌های گزارش شده مجاز بوده و مدیر سیستم می‌خواهد در بررسی بعدی از اطلاعات به‌روزشده استفاده کند.

تا اینجا نحوه‌ی کار ابزارهای بررسی صحت به‌صورت کلی توضیح داده شد. حال می‌خواهیم ارتباط بین اجزای این برنامه و سیستم فایل موردبررسی را بررسی کنیم.

به‌طور کلی نحوه‌ی ارتباط اجزای سیستم بررسی صحت و سیستم فایل در شکل زیر تصویر شده است:



شکل۲-۱: نحوه‌ی ارتباط اجزای سیستم بررسی صحت سیستم فایل

همان‌گونه که از شکل مشخص است، برنامه‌ی تشخیص دستکاری باید به پایگاه داده‌ی اطلاعات و سیستم فایل موردبررسی دسترسی داشته باشد. البته با استفاده از شکل ۲-۱ می‌توان در سطح پایین‌تری به ارتباطات و علل آن‌ها پی برد.

برنامه‌ی تهیه‌ی اطلاعات پرونده‌ها، همان‌طور که از نام آن مشخص است، برای تهیه‌ی لیست اسامی پرونده‌ها و مسیر دسترسی به آن‌ها، نیاز به استفاده از اطلاعات سیستم فایل دارد که درنهایت

نتیجه را (چه دایمی برای مقایسه‌ی بعدی و چه موقت برای مقایسه‌ی فعلی) در پایگاه داده ذخیره خواهد کرد.

برنامه‌ی تولید کدهای تشخیص دستکاری، با استفاده از اطلاعات تولید شده توسط برنامه‌ی تهیه‌ی اطلاعات پرونده‌ها در پایگاه داده و محتوای پرونده‌ها در سیستم فایل موردبررسی، کدهای تشخیص دستکاری مربوط به هر پرونده را محاسبه و در پایگاه داده ذخیره می‌کند.

درنهایت برنامه‌ی مقایسه و تولید گزارش، پس از مقایسه‌ی نتایج قبلی موجود در پایگاه داده و نتایج جدید حاصله، گزارشی از وضعیت صحت پرونده‌های سیستم فایل (یا بخشی از سیستم فایل که موردبررسی قرار گرفته است) را ارایه می‌دهد.

شکل ۲-۱ یک مدل کلی از نحوه‌ی ارتباط و دسترسی‌های اجزای برنامه‌های بررسی صحت سیستم فایل را نشان می‌دهد. این شکل ممکن است در پیاده سازی‌های مختلف از نظر جزئیات متفاوت باشد.

۲-۳- خلاصه‌ی پیام و الگوریتم MD5

همان‌گونه که قبلا اشاره شد، روش کارآمد داشتن یک امضای واحد برای هر پرونده، ساخت امضای با طول ثابت به‌ازای هر پرونده است. این عمل مبتنی بر توابع درهم‌سازی یک‌مرحله‌ای است[1] که یک قطعه‌ی طولانی را از متن اصلی گرفته و یک رشته‌ی بیتی با طول ثابت از آن محاسبه و

[1] - **One Way Hash Function**

استخراج می‌کند. حاصل این توابع درهم‌سازی اغلب به نام خلاصه‌ی پیام[1] با علامت اختصاری MD شناخته می‌شود و چهار ویژگی زیر را دارد: [۱]

۱. با داشتن یک قطعه متن[2]، محاسبه‌ی MD(p) بسیار ساده و کم هزینه است.

۲. با داشتن MD(p)، پیدا کردن p عملا غیرممکن است.

۳. با داشتن p مفروض نمی‌توان یک ´p پیدا کرد، به نحوی که MD(p) = MD(p´)

۴. تغییر در ورودی حتی به اندازه‌ی یک بیت، خروجی کاملا متفاوتی ایجاد خواهد کرد.

برای برآورده کردن ویژگی سوم، طول خلاصه‌ی پیام تولید شده باید حداقل ۱۲۸ بیت یا ترجیحا بیشتر باشد. برای برآورده کردن نیاز چهارم نیز رشته‌ی تولیدشده باید براساس تمام بیت‌های درهم‌شده‌ی پیام محاسبه شود. [۱]

۲-۳-۱. الگوریتم MD5 [۱]

توابع متنوعی برای تولید "خلاصه ی پیام" پیشنهاد شده است که رایج‌ترین آن‌ها MD5 (ری‌وست، ۱۹۹۲) و SHA-1 (NIST، ۱۹۹۳) است.MD5 پنجمین روش پیاده سازی "خلاصه‌ی پیام" است که همگی توسط رونالد ری وست (سرپرست گروه ابداع کننده ی RSA) طراحی شده‌اند. این روش با درهم‌فشردن تمام بیت‌ها، طبق رابطه‌ای بسیار پیچیده،MD را به‌نحوی محاسبه می‌کند که از تک‌تک بیت‌های ورودی متن اصلی تاثیر می‌پذیرند. به‌طور کاملا خلاصه، این روش ابتدا آن‌قدر بیت‌های بی‌اهمیت به متن اصلی اضافه می‌کند که طول آخرین بلوک، فقط و فقط ۴۴۸ بیت باشد.

---
[1] - Message Digest
[2] - Pattern

سپس طول واقعی پیام در یک فیلد ۶۴ بیتی به پیام اضافه می‌شود تا نهایتا تعداد صحیحی از بلوک‌های ۵۱۲ بیتی به‌دست آید. یعنی طول کل متن اصل ضریبی از ۵۱۲ شود. سپس در تکمیل مراحل پیش‌پردازش، یک بافر ۱۲۸ بیتی با یک عدد ثابت مقداردهی اولیه می‌شود.

حال محاسبات آغاز می‌شود. در هر دور از محاسبات یک بلوک ۵۱۲ بیتی از متن ورودی استخراج و طبق یک رابطه‌ی خاص با بافر ۱۲۸ بیتی ادغام (مخلوط) می‌شود. سپس یک جدول که با مقادیر تابع Sin مقداردهی شده در آن تزریق می‌شود. استفاده از تابع شناخته‌شده‌ای مثل Sin از آن جهت نیست که مقادیر آن تصادفی‌تر از مولد اعداد تصادفی است، بلکه فقط کمک می‌کند که جلوی هرگونه سوءظن درخصوص طراحی یک رخنه‌ی پنهان (درب پشتی) درون الگوریتم گرفته شود. ری‌وست (مخترع MD-5) درحقیقت می‌خواسته از هرگونه شایعه جلوگیری کند. در ادامه، چهار دور پردازش بر روی هر بلوک ورودی انجام می‌شود. این فرآیند آنقدر تکرار می‌شود تا محاسبه‌ی تمام بلوک‌های ورودی خاتمه یابد. محتوای بافر ۱۲۸ بیتی، خلاصه‌ی پیام را تشکیل می‌دهد.

اکنون بیش از یک دهه است که MD5 معرفی شده وبسیاری از افراد سعی در حمله به آن داشته‌اند. با وجود این‌که تعدادی نقطه ضعف در آن پیدا شده، برخی از مراحل درونی الگوریتم نگذاشته این الگوریتم شکسته شود. بااین‌وجود اگر این چند مرحله‌ی باقی‌مانده که حصارهای نهایی MD5 محسوب می‌شوند شکسته شود، MD5 فرو می‌پاشد ولی علیرغم این موضوع، تا سال ۲۰۰۳ (زمان نوشته شدن مرجع [۱])، MD5 هنوز هم به‌طور گسترده ای رایج است. (البته تا زمان نوشته شدن

این پایان‌نامه نیز هنوز MD5 به‌طور گسترده‌ای مورداستفاده قرار دارد و هنوز امنیت استفاده از این الگوریتم فروپاشیده نشده است.)

الگوریتم MD5 در RFC 1321 به‌صورت استاندارد درآمده است. برای آگاهی بیش‌تر از نحوه‌ی کار این الگوریتم و پیاده‌سازی آن، می‌توانید در پیوست الف این استاندارد را مشاهده کنید.

فصل سوم:

بررسی محدودیت‌ها و مشکلات

در فصل‌های قبلی، به معرفی بررسی‌کننده‌های صحت پرونده‌ها و نحوه‌ی کار آن‌ها پرداخته شد. همچنین الگوریتم MD5 به‌صورت اجمالی معرفی شد. همان‌گونه که گفته شد، ابزارهای بررسی صحت پرونده‌ها، برای تشخیص دستکاری غیرمجاز پرونده‌ها به‌کار می‌روند. اما ساختار، نحوه‌ی عملکرد و پیاده‌سازی این ابزارها مشکلاتی دارد که برای عملکرد درست و یا کاربردی مفیدتر باید حل شوند. در این فصل این مشکلات را معرفی خواهیم کرد:

۳-۱- تقلید امضا[1]

با توجه به بحث‌های پیشین، برای ساخت یک امضای واحد، از توابع درهم‌سازی برای تولید یک کد تشخیص دستکاری با طول ثابت استفاده می‌شود. یکی از نتایج استفاده از امضاهای با طول ثابت نگاشت چندتایی[2] است، به این معنی که: برای هر امضای تولیدشده از یک پرونده، پرونده‌های بسیاری با طول مختلف وجود دارند که همین امضا از آن‌ها به دست می‌آید. [۵]

مهاجمان می‌توانند یک پرونده را طوری تغییر دهند که امضای تولید شده از آن پس از تغییر، همانند امضای پرونده‌ی اصلی باشد و درنتیجه در بررسی صحت آن پرونده، تشخیص دستکاری ممکن نخواهد بود. یکی از روش‌هایی که برای این منظور می‌تواند به‌کار رود، روش تولید و آزمون[3] است. در این روش، مهاجم به‌صورت متناوب پرونده‌ی موردنظر را تغییر می‌دهد و کد تشخیص دستکاری را

[1] - Signature Spoofing
[2] - Multiple Mapping
[3] - Brute Force

پس از هر بار تغییر تولید می‌کند. برای یک امضا با طول n بیت، مهاجم حداکثر باید 2^{n-1} بار برای یافتن تصادم امضا[1] تلاش کند. [۵]

عمل تولید و آزمون به‌وسیله‌ی یک ایستگاه کاری همه منظوره، درصورتیکه طول امضای تولیدشده کوتاه باشد، زمان زیادی لازم نخواهد داشت. برای مثال به‌منظور یافتن یک تصادم امضا برای پرونده Login در مسیر /bin/ از سیستم عامل SunOS 4.1 (که طول پرونده ۴۷ kB است)، اگر از یک ایستگاه کاری Sparc 1+ (ماشین ۱۲/۵ MIPS) استفاده شود و از تابع CRC[2] شانزده بیتی برای تولید امضا استفاده شده باشد، مدت زمان ۰/۴۲ ثانیه برای یافتن یک تصادم نیاز خواهد بود. اگر از تابع CRC سی و دو بیتی استفاده شود، عمل یافتن تصادم امضا حدود ۴ ساعت به طول خواهد انجامید. [۵]

یکی دیگر از روش‌هایی که برای یافتن پرونده‌های متفاوت با امضای یکسان استفاده می‌شود، عمل مهندسی معکوس تابع درهم‌سازی است. اگر شخصی با آگاهی از نحوه‌ی کار تابع تولید امضا بتواند به وسیله‌ی مهندسی معکوس رخنه‌ای در کارکرد تابع مذکور بیابد، خواهد توانست تابع درهم سازی را معکوس و درنتیجه تعداد دلخواهی پرونده با همان امضای پرونده‌ی اصلی تولید کند. [۷]

به همین دلایل الگوریتم‌های تولید خلاصه‌ی پیغام در بررسی صحت اهمیت زیادی دارند. غالبا طول امضای تولیدشده باید به اندازه‌ی کافی بزرگ باشد که اغلب حداقل ۱۲۸ بیت برای آن

---

1 - **Signature Collision**

2 - **CRC مخفف عبارت Cyclic Redundancy Check code است. این تابع یکی از توابع درهم سازی برای تولید امضا یا همان کد تشخیص دستکاری است.**

درنظر گرفته می‌شود. از طرفی پیچیدگی تابع درهم سازی باید به اندازه‌ای باشد که معکوس‌پذیری آن ممکن نباشد.

جدول شماره ۳-۱ تعداد دفعات تصادم امضا برای ۲۵۴۶۸۶ پرونده را برای الگوریتم‌های مختلف نشان می‌دهد. این پرونده‌ها از مسیر کاری کاربران یک سیستم در یک محیط آزمایشگاهی به‌صورت تصادفی برداشته شده اند و درنتیجه لزوما منطقی بین محتوای آن‌ها وجود ندارد. در این جدول الگوریتم‌های مختلف مشاهده می‌شوند که در این آزمایش نتایج متفاوتی دارند. [۵]

Frequency of Signature Collisions
(254,686 signatures)

Signature	Number of collisions									Total
	1	2	3	4	5	6	7	8	>9	
16-bit checksum (sum)	14177	6647	2437	800	235	62	12	2	1	24375
16-bit CRC	15022	6769	2387	677	164	33	5	0	0	25059
32-bit CRC	3	1	1	0	0	0	0	0	0	5
64-bit DES-CBC	1	1	0	0	0	0	0	0	0	2
128-bit MD4	0	0	0	0	0	0	0	0	0	0
128-bit MD5	0	0	0	0	0	0	0	0	0	0
128-bit Snefru	0	0	0	0	0	0	0	0	0	0

جدول ۳-۱: تعداد دفعات تصادم امضا برای الگوریتم‌های مختلف

با توجه به جدول ۳-۱ و مطالبی که در بخش ۲-۳ مورد بحث قرار گرفت، می‌بینیم که الگوریتم MD5 با امضای تولیدی به طول ۱۲۸ بیت یکی از الگوریتم‌های مناسبی است که می‌تواند برای ابزارهای تشخیص دستکاری و صحت پرونده‌ها به‌کار رود. (البته مباحث مربوط به میزان

پیچیدگی الگوریتم، چون خارج از حوصله‌ی بحث است، بررسی نشده است. درهرحال الگوریتم MD5 امروزه بیش از سایر الگوریتم‌ها در ابزارهای بررسی صحت مورداستفاده قرار می‌گیرد.)

۳-۲- تشخیص دستکاری پس از ایجاد تغییرات

بررسی صحت پرونده‌ها و تشخیص دستکاری‌های غیرمجاز از نیازمندی‌های مدیران سیستم‌هاست که بتوانند تغییراتی را که در سیستم ایجاد شده تشخیص دهند. در بخش ۲-۲ که در آن به نحوه‌ی عملکرد ICها پرداخته شد، گفتیم که ابزارهای بررسی صحت باید به‌صورت دوره‌ای اجرا شوند تا تغییرات ایجادشده را تشخیص دهند. البته مدیران سیستم غالبا برای عمل برنامه‌ریزی زمانی می‌کنند و بررسی صحت به‌صورت دوره‌ای و خودکار در فواصل زمانی مشخص و معقولی صورت می‌پذیرد.

مشکلی که این‌جا مطرح است این است که چنین سیستمی در تشخیص حملاتی که در بین دو فاصله‌ی زمانی اجرای IC اتفاق می‌افتند، به اندازه‌ی کافی مؤثر عمل نمی‌کند. اگر بتوان تشخیص دستکاری را در زمان دستکاری انجام داد، درآن‌صورت می‌توان یک روال کنترلی مناسب روی سیستم اعمال کرد. [۸]

یکی از راه‌هایی که برای این منظور در مرجع [۸] مطرح شده، اجرای برنامه‌ی بررسی صحت در سطح سیستم فایل است. در این روش یک رابط در سطح هسته‌ی سیستم عامل بر روی سیستم فایل اصلی (برای مثال Ext3 برای سیستم عامل لینوکس) می‌نشیند و تمامی درخواست‌های مربوط به سیستم

فایل از آن می‌گذرد. حال در صورت ایجاد تغییرات بدون مجوز، در همان لحظه‌ی اول تغییر، دستکاری غیرمجاز تشخیص داده شده و به مدیر سیستم اخطار لازم داده می‌شود. از طرف دیگر طبق سیاست‌های پیکربندی‌شده‌ی اولیه، دسترسی به برخی منابع (مانند پرونده‌ی تغییرداده‌شده یا برنامه‌ای که تغییرات را ایجاد می‌کند) مسدود می‌شود. (برای اطلاعات بیشتر در رابطه با این مبحث می‌توانید به مرجع [۸] مراجعه کنید.)

۳-۳- عدم استقلال برنامه‌ی تشخیص دستکاری از سیستم عامل

ابزارهای امنیتی برای اجرا نباید به برنامه‌ی دیگری نیاز داشته و باید تمام اجزای مورد نیاز خود را با خود داشته باشند. برای مثال اگر یک بررسی‌کننده‌ی صحت از برنامه‌ی diff[۱] استفاده کند، درصورتی‌که این برنامه توسط مهاجم تغییر داده شود، بررسی کننده صحت به درستی کار نخواهد کرد. بنابراین برنامه‌ی تشخیص صحت باید تمامی اجزای خود را خود پیاده‌سازی کند و به سیستم وابستگی نداشته باشد. [۵]

در اینجا دو مشکل وجود دارد که برای صحت کارکرد یک برنامه‌ی تشخیص دستکاری مطرح می‌شوند. مشکل اول بحثی است که در بخش ۱-۱-۳ مطرح شد. اگر تمامی اجزای برنامه‌ی تشخیص صحت جدا از سیستم پیاده‌سازی شود، درنهایت در سطحی از اجرا به فراخوانی‌های سیستمی[۲] نیاز خواهد بود تا دسترسی به منابع سخت‌افزاری حاصل شود. حال اگر رابط‌های فراخوانی سیستمی یا

---

[1] - برنامه‌ای در سیستم عامل لینوکس که تفاوت دو پرونده را بررسی می‌کند و گزارش می‌دهد.

[2]- System Calls

خود هسته‌ی سیستم عامل مورد تغییر واقع شده باشند، دیگر نمی‌توان به برنامه‌ی تشخیص صحت اعتماد کرد. در بخش ۱-۱-۳ دیدیم که یک Rootkit سطح هسته می‌تواند این عمل را انجام دهد.

مشکل دوم، دسترس‌پذیری اجزای خود برنامه‌ی تشخیص صحت است. درهرصورت مهاجمی که با سطح دسترسی بالا به سیستم نفوذ کرده، می‌تواند خود برنامه‌ی تشخیص دستکاری را تغییر دهد. البته اگر این برنامه متن-باز[1] نباشد، در ابتدا باید خود برنامه را بررسی کند تا به اجزای آن آشنا شود و پس از آن می‌تواند اجزای برنامه را به‌گونه‌ای تغییر دهد که برنامه‌ی تشخیص صحت، خروجی مورد نظر مهاجم را تولید کند؛ و اگر برنامه به‌صورت متن-باز باشد ایجاد تغییرات در برنامه‌ی تشخیص صحت بسیار ساده خواهد بود.

برای حل مشکل دوم، مرجع [۸] ادعا می‌کند زمانی‌که ابزار شما به هسته اضافه شود و جزء ماژول‌های هسته شود، از این تغییر مصون خواهد بود. البته این عمل تا حدود زیادی از تغییر اجزای برنامه‌هایی که در وضعیت کاربری[2] هستند جلوگیری می‌کند، ولی در سیستم عامل‌های متن-باز (مانند لینوکس) که به‌راحتی به اجزای هسته‌ی سیستم عامل دسترسی وجود دارد، خود هسته‌ی سیستم عامل قابل تغییر خواهد بود.

برای حل مشکل اول، نکته‌ای که وجود دارد این است که درهرصورت برای هر نوع دسترسی به منابع باید از سیستم عامل کمک گرفت. برای این‌که از فراخوانی‌های سیستمی همان سیستمی که مورد بررسی صحت است استفاده نشود، باید به فکر چاره‌ای بود که برنامه مستقل از خود سیستم عامل

---

[1] - Open Source
[2] - User mode

کار کند. در این پروژه راه‌حلی برای این مسأله ارایه شده است. برای آشنایی با راه‌حل می‌توانید به بخش ۴-۴ مراجعه کنید. در راه‌حلی که ارایه شده است مشکل دوم نیز به‌صورت ضمنی حل می‌شود.

۳-۴- نحوه‌ی ذخیره‌ی اطلاعات در پایگاه داده

با توجه به نحوه‌ی عملکرد برنامه‌های تشخیص دستکاری که پیش‌تر به آن اشاره شد، امنیت پایگاه داده‌ی برنامه‌ی تشخیص صحت اهمیت زیادی دارد، چراکه همه‌ی امضاها و مشخصات پرونده‌ها در پایگاه داده ذخیره می‌شوند. به همین دلیل، این پایگاه داده باید از دسترسی غیرمجاز محافظت شود. اگر مهاجمی بتواند پایگاه داده را تغییر دهد، می‌تواند خروجی برنامه را به‌نفع خود تغییر دهد. یکی از راه‌هایی که برای جلوگیری از ایجاد تغییرات در پایگاه داده به‌کار می‌رود، ذخیره‌ی پایگاه داده بر روی یک رسانه‌ی فقط-خواندنی[1] است. برای مثال اگر پایگاه داده بر روی یک لوح فشرده ذخیره شود، امکان تغییر در آن وجود نخواهد داشت. [۵]

استفاده از رسانه ی فقط-خواندنی از لحاظ امنیتی راه‌حل خوبی است، ولی ازنظر کارکردی باعث ایجاد برخی مشکلات می‌شود. در مواقعی مدیر سیستم نیاز دارد پایگاه داده را به‌روز کند و ذخیره‌ی پایگاه داده روی رسانه‌ی فقط-خواندنی، مکانیزم به‌روزرسانی را کمی مشکل و کند می‌کند.[۵]

---

[1] - Read only

آن‌چه مسلم است، باید در تغییر و دسترسی به پایگاه داده محدودیت وجود داشته باشد و باید راه‌حلی برای آن درنظر گرفته شود که پایگاه داده تاحدامکان قابل تغییر نباشد و اگر مهاجم توانست تغییری ایجاد کند، نتواند از این تغییر به نفع خود استفاده کند. در بخش ۴-۵ راجع به این مسئله بحث خواهد شد.

فصل چهارم:

راه‌حل و ساختار پیشنهادی

در این فصل برای مشکلات مطرح شده در بخش ۳-۳ و ۳-۴، راه‌حل پیشنهادی ارایه خواهد شد. اما برای درک ساده‌تر این موضوع، قبلا برخی مباحث مقدماتی مطرح خواهند گشت. در ابتدا نیز محدوده‌ی پروژه‌ی انجام‌شده که موضوع آن عنوان همین پایان‌نامه است، مشخص خواهد شد.

۴-۱- محدوده‌ی پروژه

در این پروژه (با توجه به عنوان آن) یک برنامه‌ی تشخیص دستکاری برای پرونده‌های اجرایی[1] سیستم عامل لینوکس پیاده‌سازی شده است. در سیستم‌های عامل لینوکس مجوزهای دسترسی یک پرونده برای سه دسته از کاربران به‌صورت جداگانه اعطا می‌شود. دسته‌ی اول مالک پرونده[2] است، دسته‌ی دوم گروه[3] تعریف شده برای پرونده، و دسته‌ی آخر بقیه‌ی کاربران[4] (همه، بجز مالک پرونده و گروه تعریف‌شده) هستند. مجوزهای دسترسی نیز در سیستم عامل لینوکس به سه دسته تقسیم می‌شوند:

۱. مجوز خواندن پرونده که با r مشخص می‌شود.

۲. مجوز تغییر یا حذف پرونده که با w مشخص می‌شود.

۳. مجوز اجرای پرونده که با x مشخص می‌شود.

[1] -Executable
[2] - Owner (User)
[3] - Group
[4] - Other

به‌صورت کلی، اعطای هر ترکیبی از مجوزهای دسترسی به هر ترکیبی از سه دسته‌ی کاربران (توسط کاربر مجاز) امکان پذیر است. منظور از پرونده‌ی اجرایی در این پروژه، پرونده‌ای است که به هر ترکیب از مالک، گروه تعریف‌شده یا دیگر کاربران آن پرونده، دسترسی x داده شده است.

این پروژه برای بررسی صحت سیستم فایل در سیستم‌های عامل لینوکس طراحی شده است. برنامه‌ی نوشته شده بر روی سیستم‌های عامل اوبانتو[1] و فدورا-۴[2] تست شده است.

۴-۲- معرفی سیستم عامل اوبانتو [۹]

ازآن‌جاکه بستر انتخاب‌شده برای برنامه‌ی نوشته شده در این پروژه، سیستم عامل اوبانتو است، در این بخش به معرفی این سیستم عامل می‌پردازیم.

لغت اوبانتو از یک زبان باستانی آفریقایی گرفته شده و به معنی "بشریت برای همه[3]" است. اوبانتو یک سیستم عامل متن-باز[4] و غیرتجاری[5] است که بر پایه‌ی سیستم عامل دبین[6] نوشته شده و هر شش ماه یک بار نسخه جدید خود را ارایه می‌دهد. تمرکز اساسی در پیاده‌سازی اوبانتو روی راحتی استفاده‌ی کاربر آن است. اوبانتو از به‌روزرسانی امنیتی سیستم عامل نسخه‌ی رومیزی[7] خود به‌مدت سه سال، و نسخه خدمت‌گزار[8] به مدت پنج سال پشتیبانی می‌کند.

---

[1] - Ubuntu
[2] - Fedora Core 4
[3] - Humanity to Others
[4] - Open Source
[5] - Free
[6] - Debian
[7] - Desktop Version
[8] - Server Version

سیستم عامل اوبانتو از نرم‌افزارها و برنامه‌های کابردی سیستم عامل نیز به‌خوبی پشتیبانی می‌کند. گستره‌ی زیادی از انواع برنامه‌های کاربردی موجود برای لینوکس توسط سیستم عامل اوبانتو پشتیبانی می‌شوند. تمامی این برنامه‌ها به‌صورت غیرتجاری و آزاد از سایت‌های اینترنتی اوبانتو قابل دسترسی، نصب و استفاده هستند.

سیستم عامل اوبانتو تا حد زیادی به سیستم عامل دبین شباهت دارد. در حقیقت اوبانتو بر پایه‌ی معماری و ساختار دبین نوشته شده است، البته تفاوت‌هایی در نحوه‌ی ساخت و ارایه‌ی نسخه جدید خود دارد. (برای کسب اطلاعات بیشتر راجع به اوبانتو و دبین، می‌توانید به مرجع [۹] مراجعه کنید.)

۴-۳- لوح فشرده‌ی زنده[1] [۱۰]

لوح فشرده‌ی زنده، یک نوع پیاده‌سازی از سیستم عامل است که در آن عموما ماژول‌های سیستم عامل روی یک رسانه‌ی[2] قابل راه‌اندازی[3] مانند لوح فشرده قرار داده می‌شود که به آن "لوح فشرده‌ی زنده" می‌گویند. این سیستم عامل بدون نیاز به نصب روی دیسک سخت[4]، از روی لوح فشرده قابل اجرا است. لغت "زنده" نیز به‌این‌دلیل انتخاب شده است که این پیاده‌سازی برای اجرا به دیسک سخت سیستم وابستگی ندارد و خود می‌تواند اجرا شود.

[1] - Live CD
[2] - Media
[3] - Bootable
[4] - Hard Disk

پس از اجرای سیستم عامل از روی لوح فشرده، برای بازگشتن به حالت قبلی و اجرای سیستم عامل موجود بر روی دیسک سخت، کافی‌ست که سیستم را راه‌اندازی مجدد[1] و لوح فشرده را از آن خارج کرد.

غالبا سیستم عامل لوح فشرده زنده، نسبت به همان سیستم عامل وقتی‌که از روی دیسک سخت در حال اجراست، کارآیی پایین‌تری دارد. یکی از دلایل این موضوع، سرعت دسترسی پایین‌تر به لوح فشرده نسبت به دیسک سخت است. دلیل دیگر این است که لوح فشرده‌ی زنده مانند دیسک سخت، فضایی برای حافظه‌ی مجازی سیستم عامل ندارد. درنتیجه به‌ناچار باید از فضای حافظه‌ی اصلی برای حافظه مجازی[2] نیز (البته به‌صورت محدودتر) استفاده کند.

امروزه سیستم‌های عامل بسیاری وجود دارند که پیاده سازی لوح فشرده زنده را نیز ارایه داده‌اند، که برای مشاهده‌ی لیست آن‌ها می‌توانید به مرجع [۱۰] مراجه کنید. از جمله‌ی این سیستم‌های عامل می‌توان به سیستم عامل اوبانتو اشاره کرد که نسخه‌ی لوح فشرده‌ی زنده‌ی خود را همراه با نسخه‌ی قابل‌نصب در هر بار انتشار نسخه‌ی جدید ارایه می‌دهد.

ابزار آماده‌شده در این پروژه درنهایت برروی لوح فشرده‌ی زنده‌ی سیستم عامل اوبانتو قرار می‌گیرد. علت این امر در بخش‌های آتی توضیح داده شده است.

---
[1] - Reboot
[2] - Virtual Memory

۴-۴- استقلال برنامه‌ی تشخیص دستکاری از سیستم عامل

در بخش ۳-۳ به دو مشکل از مشکلات برنامه‌های تشخیص دستکاری اشاره شد. این دو مشکل، از عدم استقلال برنامه‌ی تشخیص دستکاری از سیستم فایل و سیستم عامل (به‌طور کلی سیستمی که در حال بررسی است) ناشی می‌شوند. در آن بخش گفته شد که باید به فکر چاره‌ای بود تا از خود سیستم درحال بررسی به‌هیچ‌عنوان برای بررسی صحت استفاده نشود؛ حتی در سطح فراخوانی‌های سیستمی و استفاده از هسته‌ی سیستم عامل برای دسترسی به منابع.

برای رسیدن به این منظور، باید به‌گونه‌ای خود سیستم عامل در حال اجرا نباشد؛ چون درغیراین‌صورت سیستم عامل مدیریت تمامی منابع را به عهده خواهد گرفت و اسقلال از سیستم عامل بی‌معنی خواهد بود. ازطرفی درصورتی‌که سیستم عامل درحال اجرا نباشد، امکان بررسی صحت (یا هر عمل دیگری) عملا بی‌معنی و غیرممکن خواهد بود.

راه‌حلی که وجود دارد این است که سیستم عامل موردبررسی درحال اجرا نباشد و سیستم فایل موردبررسی توسط یک سیستم عامل دیگر مورد بررسی قرار گیرد. امکان نصب یک سیستم عامل جدید نیز وجود ندارد؛ چراکه هدف، بررسی صحت پرونده‌های موجود است و اگر بخواهیم این پرونده‌ها را خود حذف یا دستکاری نماییم، مسئله حل نشده است. برای حل این مسئله، برنامه‌ی تشخیص صحت را بر روی لوح فشرده‌ی زنده‌ی سیستم عامل اوبانتو قرار داده‌ایم تا درصورت نیاز به بررسی صحت، سیستم عامل مورد بررسی را به حالت خاموش در آورده و توسط لوح فشرده‌ی زنده

سیستم را راه‌اندازی کنیم. پس از آن می‌توان توسط این لوح فشرده، سیستم فایل حافظه‌ی دیسک سخت را مورد بررسی قرار داد.

همان‌طور که در بخش ۴-۳ دیده شد، زمانی‌که سیستمی توسط لوح فشرده‌ی زنده راه‌اندازی می‌شود، سیستم عامل موجود در لوح فشرده است که به اجرا در می‌آید و کنترل تمامی منابع را دراختیار می‌گیرد. همه‌ی برنامه‌هایی که ازروی لوح فشرده اجرا می‌شوند، مستقل از سیستم عامل موجود روی حافظه ی دیسک سخت هستند. درنتیجه همه‌ی اعمال برنامه‌ی تشخیص دستکاری نیز مستقل از سیستم عامل و فراخوانی‌های سیستمی سیستم مورد بررسی خواهد بود. ازطرفی به دلیل این‌که لوح فشرده‌ی زنده، یک رسانه‌ی فقط-خواندنی است، مسئله‌ی دستکاری اجزای برنامه‌ی تشخیص صحت نیز منتفی است و این مشکل برای این راه‌حل دیگر مطرح نیست.

تا کنون یک روش پیشنهادی برای حل دو مشکل مطرح شده در بخش ۳-۳ ارایه شد، اما این روش معایبی هم دارد. مورد اول، خاموش شدن سیستم مورد بررسی است. امکان خاموش کردن بسیاری از سیستم‌ها به دلیل سرویس‌هایی که آن‌ها ارایه می‌دهند، وجود ندارد. یکی از دلایل آن، که به سیاست‌های تجاری یا امنیتی یک سازمان مرتبط است، بحث دسترسی[1] سرویس‌ها و تداوم کسب‌وکار آن‌هاست. سیاست‌های تجاری ممکن است بتوانند سطحی از عدم ایمنی را بپذیرند ولی نتوانند سرویس خود را برای مدتی (به‌صورت تجربی در حدود یک ساعت) ارایه ندهند. در این صورت است که ممکن است روش ارایه شده در این پروژه قابل قبول نباشد.

---

[1] - **Availability**

مورد دوم، بحث کارآمدی برنامه است. غالبا اجرای برنامه‌ها از روی حافظه ی دیسک سخت سرعت بالاتری نسبت به اجرا از روی لوح فشرده دارد. دلیل آن نیز در بخش ۴-۳ توضیح داده شد.

مورد سوم، نحوه‌ی ذخیره‌سازی اطلاعات در پایگاه داده است. فرض ما این است که به‌هیچ‌عنوان به سیستم فایل مورد بررسی نیاز نداشته باشیم. از طرفی امکان ذخیره‌ی اطلاعات روی لوح فشرده‌ی زنده وجود ندارد. پس باید به‌دنبال راه‌حلی برای این مشکل بود. در بخش بعدی این مسئله مورد بررسی قرار می‌گیرد.

۴-۵- مسایل مربوط به پایگاه داده

مسئله‌ی اول، بحث مربوط به عدم تغییر اطلاعات پایگاه داده توسط مهاجم است. پیشتر گفته‌شد که مهاجم تا جای ممکن نباید بتواند اطلاعات پایگاه داده را تغییر دهد. اما درهرصورت، درصورت تغییر دادن، این تغییر باید تشخیص داده شود. مسئله دیگری نیز در بخش قبل مطرح شد و آن این بود که پایگاه داده‌ی ساختار پیشنهادی نمی‌تواند روی سیستم مورد بررسی یا لوح فشرده باشد.

راه‌حل ممکن استفاده از پایگاه داده‌ی موجود روی سیستمی دیگر در شبکه است. درهرصورت اطلاعات مورد نظر باید روی سیستم سومی به‌جز لوح فشرده‌ی زنده و سیستم موردبررسی ذخیره شود.

پیشنهاد می‌شود که پس از اتمام اجرای برنامه، از اطلاعات پایگاه داده، نسخه‌ی پشتیبانی روی رسانه‌ی فقط-خواندنی تهیه شود و به‌عنوان منبع اطلاعات در بررسی‌های بعدی مورداستفاده قرار گیرد.

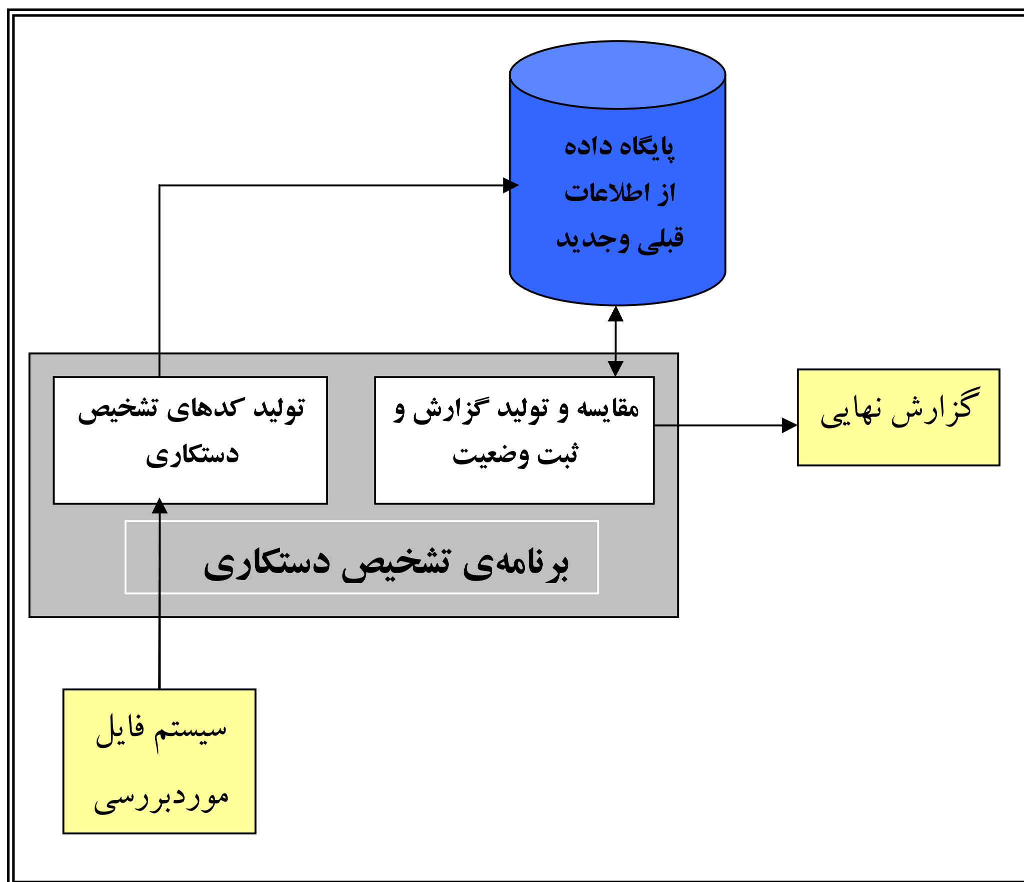
اما با فرض انجام اعمال بالا، اگر اطلاعات بتوانند بر روی سیستم سوم تغییر یابند، امکان سوءاستفاده را به مهاجمان خواهند داد. پس باید به دنبال راهی برای تشخیص دستکاری اطلاعات پایگاه داده باشیم، یا کاری کنیم که مهاجم نتواند پرونده‌ای را تغییر دهد و امضای آن را (که الگوریتم تولید امضا مشخص است) تولید و در پایگاه داده جایگزین کند.

برای رسیدن به این هدف، هنگام تولید امضا، از مدیر سیستم که در حال اجرای برنامه‌ی تشخیص صحت است، یک کلمه‌ی رمز درخواست می‌شود و این کلمه در مراحل تولید امضا، بر امضای نهایی تأثیر می‌گذارد. نحوه‌ی کار به‌این‌صورت است که پس از تولید امضای پرونده (که رشته‌ای از کاراکترها است) از محتوای پرونده، امضای موردنظر با کلمه‌ی رمز کاربر و برخی مشخصات پرونده ترکیب شده و رشته‌ی به‌دست‌آمده دوباره برای تولید امضا به ورودی برنامه‌ی تولید امضا داده می‌شود. دراین‌صورت درنهایت امضای تولیدشده برای هر پرونده، هم به مشخصات آن پرونده و هم به کلمه‌ی رمز موردنظر مدیر سیستم وابسته است. حال اگر مهاجم پرونده‌ای را تغییر دهد و امضای جدید تولید کند، به دلیل نداشتن کلمه‌ی رمز، نمی‌تواند امضای نهایی را تولید کند. البته درنهایت مهاجم می‌تواند صحت برنامه‌ی تشخیص صحت را به خطر بیاندازد. اگر مهاجم به پایگاه داده دسترسی داشته باشد، می‌تواند یک امضای موجود در پایگاه داده را با امضای دیگری از پایگاه داده جایگزین کند. درهرصورت امنیت پایگاه داده‌ها در بحث برنامه‌های تشخیص صحت مورد بررسی قرار نمی‌گیرد و فرض، داشتن یک پایگاه داده‌ی امن است. اگرچه تا حدی که به برنامه‌ی تشخیص صحت مربوط باشد و این برنامه بر امنیت پایگاه داده تأثیر بگذارد، به این حوزه وارد شده و راه‌کارهایی عنوان می‌شود.

۴-۶- نحوه‌ی کار برنامه‌ی تشخیص دستکاری پرونده‌های اجرایی[1]

۴-۶-۱. معماری سیستم

اجزای برنامه‌ی تشخیص دستکاری،کارها و ارتباط آن‌ها در شکل زیر آورده شده است:



شکل ۴-۱: اجزای برنامه‌ی تشخیص دستکاری پرونده‌های اجرایی و ارتباط‌های آن‌ها

همان‌گونه که در این شکل مشاهده می‌شود، برنامه‌ی تشخیص دستکاری از دو بخش اصلی تشکیل شده است. بخش اول، بخش تولید کدهای تشخیص دستکاری است. این بخش وظیفه دارد

[1] - Executables Integrity Checker (EIC)

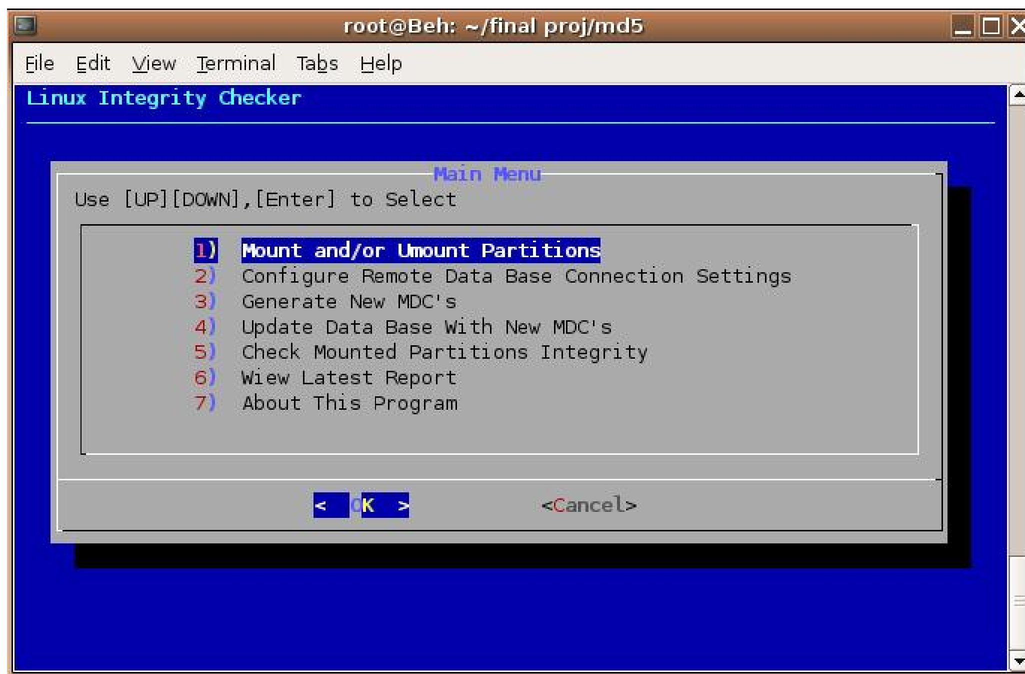
همه‌ی پرونده‌های اجرایی سیستم فایل موردبررسی را لیست، برای هر یک کد تشخیص دستکاری تولید، و سپس کدهای تولید شده را به‌همراه نام و مسیر پرونده در جدولی در پایگاه داده ذخیره کند. ازطرفی زمانی‌که برای بار بعدی به تولید کد تشخیص دستکاری برای مقایسه نیاز باشد، این بخش دوباره کدها را تولید و در جدول دیگری ذخیره می‌کند.

بخش دوم، بخش مقایسه و تولید گزارش است. این بخش در زمان بررسی صحت و تشخیص دستکاری استفاده می‌شود. این بخش نتایج موجود در دو جدول پایگاه داده را مقایسه کرده و نتیجه را در گزارش نهایی می‌نویسد. درصورتی‌که پس از گزارش‌گیری، تغییرات ایجاد شده در سیستم غیرمجاز نباشند، مدیر سیستم می‌تواند کدهای تشخیص دستکاری جدید را به‌عنوان مبنای اطلاعات برای بررسی‌های بعدی در پایگاه داده ذخیره کند تا در گزارش‌گیری آینده، این تغییرات دوباره گزارش نشوند.

۴-۶-۲. نحوه‌ی کار برنامه

بخش اصلی برنامه‌ی تشخیص دستکاری به زبان برنامه نویسی C نوشته شده است و برای نوشتن رابط کاربری از زبان متنی پوسته‌ی[1] لینوکس استفاده شده است. رابط کاربری برنامه، به‌صورت گرافیک متنی و توسط ماوس نیز قابل استفاده است. در زیر، شکل صفحه‌ی اصلی اجرای برنامه را مشاهده می‌کنید.

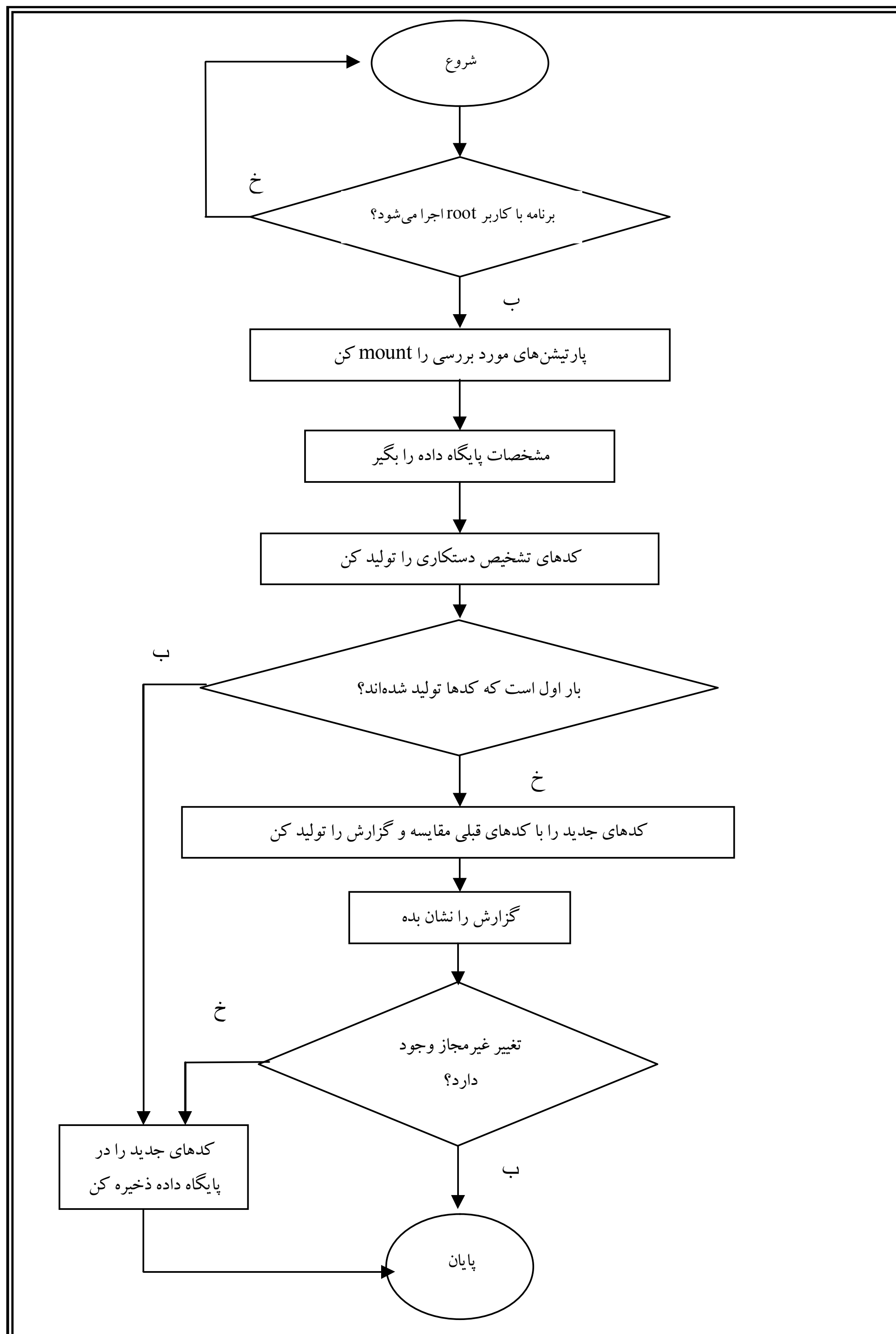
[1] - Shell Script



شکل ۴-۲ : صفحه‌ی اصلی اجرای برنامه‌ی تشخیص دستکاری

روند اجرای برنامه نیز در فلوچارت[1] زیر مشخص است:

[1] - Flow Chart



شکل ۴-۳: فلوچارت روند اجرای برنامه‌ی تشخیص دستکاری

۴-۶-۳. نحوه‌ی استفاده از برنامه

همان‌طور که در بخش‌های پیشین اشاره شده است، برای اجرای این برنامه ابتدا باید سیستم مورد بررسی را توسط لوح فشرده راه‌اندازی کرد. پس از آن‌که سیستم عامل اوبانتو از لوح فشرده اجرا شد، می‌توان برنامه‌ی تشخیص دستکاری را با اجرای دستور eic در ترمینال پوسته‌ی لینوکس اجرا کرد. پس از آن با توجه به فلوچارت بخش قبل می‌توان از برنامه استفاده کرد.

مراجع

[۱] ا. تنن بام، **شبکه‌های کامپیوتری**، ترجمه ح. پدرام، ا. ملکیان و ع. ر. زارع پور، انتشارات نص، تهران، ۱۳۸۲.

[۲] ا. اسکادیس، **راهنمای گام به گام نفوذ به کامپیوترها و راه‌های موثر مقابله با آن**، ترجمه م. توانا و س. هراتیان، انتشارات موسسه فرهنگی هنری نقش سیمرغ، تهران، ۱۳۸۱.

[۳] و. استالینگز، **سیستم‌های عامل**، ترجمه ح. پدرام و م. صدیقی مشکنانی، انتشارات شیخ بهایی، اصفهان، ۱۳۸۲.

[4] *Information Security Management Systems – Specifications With Guidance For Use*, British Standard, BS 7799-2, 2002.

[5] G. H. Kim and E. H. Spafford, “The Design and Implementation Of Tripwire: A File System Integrity Checker”, *ACM Conf. Computer and Communications Security*, 1994.

[6] V. Bontchev, “Possible Virus Attacks Against Integrity Programs And How To Prevent Them”, *Technical Report*, University of Hamburg, 1993.

[7] R. C. Merkle, “A Fast Software One-Way Hash Function”, *Journal of Cryptology*, no 3(1), 1990.

[8] “I^3FS: An In-Kernel Integrity Checker and Intrusion Detection File System”, http://www.fsl.cs.sunysb.edu, May 2006.

[9] Ubuntu, http://www.ubuntu.com, June 2006.

[10] Wikipedia, http://en.wikipedia.org, June 2006.

پیوست‌ها

پیوست الف: RFC 1321 (الگوریتم MD5)

www.sazadi.com

Network Working Group R. Rivest
Request for Comments: 1321 MIT Laboratory for Computer Science
and RSA Data Security, Inc.
April 1992

The MD5 Message-Digest Algorithm

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Acknowlegements

We would like to thank Don Coppersmith, Burt Kaliski, Ralph Merkle, David Chaum, and Noam Nisan for numerous helpful comments and suggestions.

Table of Contents

1. Executive Summary 1
2. Terminology and Notation 2
3. MD5 Algorithm Description 3
4. Summary 6
5. Differences Between MD4 and MD5 6
References 7
APPENDIX A - Reference Implementation 7
Security Considerations 21
Author's Address 21

1. Executive Summary

This document describes the MD5 message-digest algorithm. The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

www.sazadi.com Siamak Azadiabad

www.sazadi.com

The MD5 algorithm is designed to be quite fast on 32-bit machines. In addition, the MD5 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD5 algorithm is an extension of the MD4 message-digest algorithm 1,2]. MD5 is slightly slower than MD4, but is more "conservative" in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. MD5 backs off a bit, giving up a little in speed for a much greater likelihood of ultimate security. It incorporates some suggestions made by various reviewers, and contains additional optimizations. The MD5 algorithm is being placed in the public domain for review and possible adoption as a standard.

For OSI-based applications, MD5's object identifier is

```
md5 OBJECT IDENTIFIER ::=
  iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 5}
```

In the X.509 type AlgorithmIdentifier [3], the parameters for MD5 should have type NULL.

2. Terminology and Notation

In this document a "word" is a 32-bit quantity and a "byte" is an eight-bit quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group of eight bits is interpreted as a byte with the high-order (most significant) bit of each byte listed first. Similarly, a sequence of bytes can be interpreted as a sequence of 32-bit words, where each consecutive group of four bytes is interpreted as a word with the low-order (least significant) byte given first.

Let x_i denote "x sub i". If the subscript is an expression, we surround it in braces, as in x_{i+1}. Similarly, we use ^ for superscripts (exponentiation), so that x^i denotes x to the i-th power.

Let the symbol "+" denote addition of words (i.e., modulo-2^32 addition). Let X <<< s denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions. Let not(X) denote the bit-wise complement of X, and let X v Y denote the bit-wise OR of X and Y. Let X xor Y denote the bit-wise XOR of X and Y, and let XY denote the bit-wise AND of X and Y.

www.sazadi.com Siamak Azadiabad

www.sazadi.com

3. MD5 Algorithm Description

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

$$m_0 \ m_1 \ \ldots \ m_{b-1}$$

The following five steps are performed to compute the message digest of the message.

3.1 Step 1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512.

Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

3.2 Step 2. Append Length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

3.3 Step 3. Initialize MD Buffer

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
word A: 01 23 45 67
word B: 89 ab cd ef
word C: fe dc ba 98
word D: 76 54 32 10
```

3.4 Step 4. Process Message in 16-Word Blocks

We first define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XZ v Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X v not(Z))
```

In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Do the following:

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do

  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Save A as AA, B as BB, C as CC, and D as DD. */
  AA = A
  BB = B
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
     CC = C
     DD = D

     /* Round 1. */
     /* Let [abcd k s i] denote the operation
          a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
     /* Do the following 16 operations. */
     [ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
     [ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
     [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
     [ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]

     /* Round 2. */
     /* Let [abcd k s i] denote the operation
          a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
     /* Do the following 16 operations. */
     [ABCD  1  5 17]  [DABC  6  9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
     [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
     [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
     [ABCD 13  5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]

     /* Round 3. */
     /* Let [abcd k s t] denote the operation
          a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
     /* Do the following 16 operations. */
     [ABCD  5  4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
     [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
     [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
     [ABCD  9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]

     /* Round 4. */
     /* Let [abcd k s t] denote the operation
          a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
     /* Do the following 16 operations. */
     [ABCD  0  6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
     [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
     [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
     [ABCD  4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]

     /* Then perform the following additions. (That is increment each
        of the four registers by the value it had before this block
        was started.) */
     A = A + AA
     B = B + BB
     C = C + CC
     D = D + DD

   end /* of loop on i */
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com

3.5 Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

This completes the description of MD5. A reference implementation in C is given in the appendix.

4. Summary

The MD5 message-digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD5 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort.

5. Differences Between MD4 and MD5

The following are the differences between MD4 and MD5:

1. A fourth round has been added.

2. Each step now has a unique additive constant.

3. The function g in round 2 was changed from (XY v XZ v YZ) to (XZ v Y not(Z)) to make g less symmetric.

4. Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".

5. The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.

6. The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

www.sazadi.com Siamak Azadiabad

www.sazadi.com

References

[1] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1320, MIT and RSA Data Security, Inc., April 1992.

[2] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework."

APPENDIX A - Reference Implementation

This appendix contains the following files taken from RSAREF: A Cryptographic Toolkit for Privacy-Enhanced Mail:

global.h -- global header file

md5.h -- header file for MD5

md5c.c -- source code for MD5

For more information on RSAREF, send email to <rsaref@rsa.com>.

The appendix also includes the following file:

mddriver.c -- test driver for MD2, MD4 and MD5

The driver compiles for MD5 by default but can compile for MD2 or MD4 if the symbol MD is defined on the C compiler command line as 2 or 4.

The implementation is portable and should work on many different plaforms. However, it is not difficult to optimize the implementation on particular platforms, an exercise left to the reader. For example, on "little-endian" platforms where the lowest-addressed byte in a 32-bit word is the least significant and there are no alignment restrictions, the call to Decode in MD5Transform can be replaced with a typecast.

A.1 global.h

```
/* GLOBAL.H - RSAREF types and constants
 */

/* PROTOTYPES should be set to one if and only if the compiler supports
  function argument prototyping.
The following makes PROTOTYPES default to 0 if it has not already
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
  been defined with C compiler flags.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER defines a generic pointer type */
typedef unsigned char *POINTER;

/* UINT2 defines a two byte word */
typedef unsigned short int UINT2;

/* UINT4 defines a four byte word */
typedef unsigned long int UINT4;

/* PROTO_LIST is defined depending on how PROTOTYPES is defined above.
If using PROTOTYPES, then PROTO_LIST returns the list, otherwise it
  returns an empty list.
 */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

A.2 md5.h

```
/* MD5.H - header file for MD5C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.
```

www.sazadi.com


```
These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* MD5 context. */
typedef struct {
  UINT4 state[4];                                   /* state (ABCD) */
  UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
  unsigned char buffer[64];                         /* input buffer */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));
```

A.3 md5c.c

```
/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#include "global.h"
#include "md5.h"

/* Constants for MD5Transform routine.
 */
```


www.sazadi.com Siamak Azadiabad

```
#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
```

www.sazadi.com


```
 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                        /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                        /* context */
unsigned char *input;                                /* input block */
unsigned int inputLen;                     /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits */
  if ((context->count[0] += ((UINT4)inputLen << 3))
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
   < ((UINT4)inputLen << 3))
 context->count[1]++;
  context->count[1] += ((UINT4)inputLen >> 29);

  partLen = 64 - index;

  /* Transform as many times as possible.
*/
  if (inputLen >= partLen) {
 MD5_memcpy
   ((POINTER)&context->buffer[index], (POINTER)input, partLen);
 MD5Transform (context->state, context->buffer);

 for (i = partLen; i + 63 < inputLen; i += 64)
   MD5Transform (context->state, &input[i]);

 index = 0;
  }
  else
 i = 0;

  /* Buffer remaining input */
  MD5_memcpy
 ((POINTER)&context->buffer[index], (POINTER)&input[i],
  inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
  the message digest and zeroizing the context.
 */
void MD5Final (digest, context)
unsigned char digest[16];                         /* message digest */
MD5_CTX *context;                                       /* context */
{
  unsigned char bits[8];
  unsigned int index, padLen;

  /* Save number of bits */
  Encode (bits, context->count, 8);

  /* Pad out to 56 mod 64.
*/
  index = (unsigned int)((context->count[0] >> 3) & 0x3f);
  padLen = (index < 56) ? (56 - index) : (120 - index);
  MD5Update (context, PADDING, padLen);

  /* Append length (before padding) */
  MD5Update (context, bits, 8);
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
  /* Store state in digest */
  Encode (digest, context->state, 16);

  /* Zeroize sensitive information.
*/
  MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Round 1 */
  FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Round 2 */
  GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
  GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
  GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
  GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
  GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
  GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
  GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
  GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
  GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
  GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
  GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
  GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
  GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
  GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
  GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
  GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

  /* Round 3 */
  HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
  HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
  HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
  HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
  HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
  HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
  HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
  HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
  HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
  HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
  HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
  HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
  HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
  HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
  HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
  HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

  /* Round 4 */
  II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
  II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
  II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
  II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
  II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
  II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
  II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
  II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
  II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
  II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
  II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
  II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
  II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
  II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
  II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
  II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

  state[0] += a;
  state[1] += b;
  state[2] += c;
  state[3] += d;

  /* Zeroize sensitive information.
```


www.sazadi.com Siamak Azadiabad

```
*/
  MD5_memset ((POINTER)x, 0, sizeof (x));
}

/* Encodes input (UINT4) into output (unsigned char). Assumes len is
  a multiple of 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
 output[j] = (unsigned char)(input[i] & 0xff);
 output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
 output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
 output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Decodes input (unsigned char) into output (UINT4). Assumes len is
  a multiple of 4.
 */
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
```

www.sazadi.com


```
 output[i] = input[i];
}

/* Note: Replace "for loop" with standard memset if possible.
 */
static void MD5_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C - test driver for MD2, MD4 and MD5
 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Created 1990. All
rights reserved.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

/* The following makes MD default to MD5 if it has not already been
  defined with C compiler flags.
 */
#ifndef MD
#define MD MD5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4
```


www.sazadi.com Siamak Azadiabad

```
#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif

/* Length of test block, number of test blocks.
 */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif

/* Main driver.

Arguments (may be any combination):
  -sstring - digests string
  -t       - runs time trial
  -x       - runs test script
  filename - digests file
  (none)   - digests standard input
 */
int main (argc, argv)
int argc;
```

www.sazadi.com


```
char *argv[];
{
  int i;

  if (argc > 1)
 for (i = 1; i < argc; i++)
   if (argv[i][0] == '-' && argv[i][1] == 's')
     MDString (argv[i] + 2);
   else if (strcmp (argv[i], "-t") == 0)
     MDTimeTrial ();
   else if (strcmp (argv[i], "-x") == 0)
     MDTestSuite ();
   else
     MDFile (argv[i]);
  else
 MDFilter ();

  return (0);
}

/* Digests a string and prints the result.
 */
static void MDString (string)
char *string;
{
  MD_CTX context;
  unsigned char digest[16];
  unsigned int len = strlen (string);

  MDInit (&context);
  MDUpdate (&context, string, len);
  MDFinal (digest, &context);

  printf ("MD%d (\"%s\") = ", MD, string);
  MDPrint (digest);
  printf ("\n");
}

/* Measures the time to digest TEST_BLOCK_COUNT TEST_BLOCK_LEN-byte
  blocks.
 */
static void MDTimeTrial ()
{
  MD_CTX context;
  time_t endTime, startTime;
  unsigned char block[TEST_BLOCK_LEN], digest[16];
  unsigned int i;
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
  printf
 ("MD%d time trial. Digesting %d %d-byte blocks ...", MD,
  TEST_BLOCK_LEN, TEST_BLOCK_COUNT);

  /* Initialize block */
  for (i = 0; i < TEST_BLOCK_LEN; i++)
 block[i] = (unsigned char)(i & 0xff);

  /* Start timer */
  time (&startTime);

  /* Digest blocks */
  MDInit (&context);
  for (i = 0; i < TEST_BLOCK_COUNT; i++)
 MDUpdate (&context, block, TEST_BLOCK_LEN);
  MDFinal (digest, &context);

  /* Stop timer */
  time (&endTime);

  printf (" done\n");
  printf ("Digest = ");
  MDPrint (digest);
  printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
  printf
 ("Speed = %ld bytes/second\n",
  (long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}

/* Digests a reference suite of strings and prints the results.
 */
static void MDTestSuite ()
{
  printf ("MD%d test suite:\n", MD);

  MDString ("");
  MDString ("a");
  MDString ("abc");
  MDString ("message digest");
  MDString ("abcdefghijklmnopqrstuvwxyz");
  MDString
 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
  MDString
 ("1234567890123456789012345678901234567890\
1234567890123456789012345678901234567890");
}

/* Digests a file and prints the result.
```


www.sazadi.com Siamak Azadiabad

www.sazadi.com


```
 */
static void MDFile (filename)
char *filename;
{
  FILE *file;
  MD_CTX context;
  int len;
  unsigned char buffer[1024], digest[16];

  if ((file = fopen (filename, "rb")) == NULL)
 printf ("%s can't be opened\n", filename);

  else {
 MDInit (&context);
 while (len = fread (buffer, 1, 1024, file))
   MDUpdate (&context, buffer, len);
 MDFinal (digest, &context);

 fclose (file);

 printf ("MD%d (%s) = ", MD, filename);
 MDPrint (digest);
 printf ("\n");
  }
}

/* Digests the standard input and prints the result.
 */
static void MDFilter ()
{
  MD_CTX context;
  int len;
  unsigned char buffer[16], digest[16];

  MDInit (&context);
  while (len = fread (buffer, 1, 16, stdin))
 MDUpdate (&context, buffer, len);
  MDFinal (digest, &context);

  MDPrint (digest);
  printf ("\n");
}

/* Prints a message digest in hexadecimal.
 */
static void MDPrint (digest)
unsigned char digest[16];
{
```

www.sazadi.com


```
  unsigned int i;

  for (i = 0; i < 16; i++)
 printf ("%02x", digest[i]);
}
```

A.5 Test suite

The MD5 test suite (driver option "-x") should print the following results:

```
MD5 test suite:
MD5 ("") = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
d174ab98d277d9f5a5611c2c9f419d9f
MD5 ("123456789012345678901234567890123456789012345678901234567890123456
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

Security Considerations

The level of security discussed in this memo is considered to be sufficient for implementing very high security hybrid digital-signature schemes based on MD5 and a public-key cryptosystem.

Author's Address

Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986

Phone: (617) 253-5880
EMail: rivest@theory.lcs.mit.edu

www.sazadi.com Siamak Azadiabad