مجله علمی پژوهشی «علوم و فناوری‌های پدافند نوین»
سال هفتم، شماره ۱، بهار ۱۳۹۵؛ ص ۸۰-۷۱

یک چارچوب بهینه و شفاف برای تحلیل خودکار بدافزار

سعید پارسا[1*]، امیر گوران اوریمی[2]
۱- دانشیار، ۲- کارشناسی ارشد، دانشگاه علم و صنعت ایران
(دریافت: ۹۳/۱۰/۲۶، پذیرش: ۹۴/۰۲/۰۲)

چکیده

بدافزار مهم‌ترین عامل تهدیدات امنیتی در فضای مجازی است. میزان این بدافزارها به حدی است که برخی آمارها نشان می‌دهد روزانه بیش از ۳۱۵۰۰۰ بدافزار جدید منتشر می‌شود. مطمئناً تحلیل دستی این حجم از بدافزار غیرممکن است. به همین دلیل باید از نرم‌افزارهایی استفاده شود که توان تحلیل فایل‌های مشکوک را داشته و بتوانند رفتار داخلی آن را به‌طور کاملاً خودکار تعیین نمایند. تاکنون چندین ابزار همانند آنوبیس و جعبه‌شن کوکو در این زمینه ارائه شده‌اند. مهم‌ترین مشکل این ابزارها عدم شفافیت است. امروزه بدافزارهایی منتشر شده‌اند که از این مشکل موجود در ابزارهای کنونی سوءاستفاده کرده و به‌صورت چندشخصیتی عمل می‌کنند. برای حل این مشکل راهکار مجازی‌سازی به‌کمک سخت‌افزار ارائه‌شده است، اما این راه حل نیز دارای مشکل عدم بهینگی است. در این مقاله چارچوب تحلیل خودکار بدافزاری ارائه خواهد شد که شفاف و بهینه باشد. بنابراین علاوه‌بر این‌که در برابر بدافزارهای چندشخصیتی مقاوم است، توان تحلیل این میزان بدافزار منتشرشده در روز را بدون نیاز به اضافه‌کردن منابع سخت‌افزاری جدید دارد. این چارچوب از تحلیل پویا به‌همراه فناوری مجازی‌سازی به‌کمک سخت‌افزار استفاده می‌کند. در تحلیل پویا از روش‌هایی همانند جعبه شن و دنبال‌کردن فراخوانی‌های سیستمی استفاده شده است. تحلیل‌های مبتنی بر مجازی‌سازی به‌کمک سخت‌افزار نیز برای ایجاد شفافیت مورد استفاده قرار گرفته است.

کلیدواژه‌ها: محیط تحلیل، مجازی‌سازی به‌کمک سخت‌افزار، شفافیت، تحلیل پویا، بدافزار.

An Optimal and Transparent Framework for Automatic Analysis of Malware

Saeed Parsa*, A. Gooran Oorimi
Iranian University of Science and Technology
(Received: 16/01/2015, Accepted: 22/04/2015)

Abstract

Malware is the most important security threat in cyberspace. Some statistics show that over 315,000 malware are released, every day. Certainly, it is not possible to analyze all of these malware, manually. That's why the security vendors are obliged to use software capable of analyzing suspicious executable files. These software determine behavior of suspicious files automatically. Several tools such as Anubis and Cuckoo are produced in this area. The problem of these tools is lack of transparency. Some malware use this sort of weaknesses to recon analysis environments. To resolve this problem some solutions using hardware-assisted virtualization is presented. However, these solutions impose a great run time overhead on the program execution. In this paper an automated malware analysis framework is presented that is both transparent and optimal. This framework in addition to being resistant to malware with split personality features, may also be used to analyze the large amount of malware released every day without adding extra hardware resources. This framework uses dynamic analysis approaches with hardware assisted virtualization technology to analyze suspicious code. The dynamic analysis approaches used in this framework include sandboxing and system calls sequence analysis. Analysis based on hardware-assisted virtualization technology is applied to provide transparent analysis environment.

Keywords: Analysis Environment, Hardware-Assisted Virtualization, Transparency, Dynamic Analysis, Malware.

* Corresponding author Email: parsa@iust.ac.ir

۱. مقدمه

همان‌طورکه می‌دانید بدافزار امروزه به یکی از اصلی‌ترین دغدغه‌های فضای مجازی تبدیل شده است. تا سال ۲۰۰۵، روش تحلیل دستی فایل‌های مشکوک یک روش مناسب برای تعیین ماهیت آن‌ها و شناسایی بدافزار بوده است. اما با ایجاد چند موتور ساخت بدافزار چندریختی[1] و همچنین انتشار کد چند فشرده‌ساز[2] مشهور، تعداد و حجم بدافزار به‌طور گسترده افزایش یافت. بر طبق گزارش‌های شرکت مک‌آفی و کسپرسکی حجم بدافزار بین سال‌های ۲۰۰۸ تا ۲۰۱۲ افزایش بسیار زیادی داشته است [۱]. به حدی که در سال ۲۰۱۴، روزانه بالای ۳۱۵۰۰۰ بدافزار جدید منتشر می‌شود [۲]. مطمئناً تحلیل دستی این حجم از بدافزار غیرممکن است. به همین دلیل باید از نرم‌افزارهایی استفاده شود که توان تحلیل فایل‌های مشکوک را داشته و بتوانند رفتار داخلی آن را به‌طور کاملاً خودکار تعیین نمایند. تاکنون چندین ابزار همانند آنوبیس و جعبه‌شن کوکو در این زمینه ارائه شده‌اند [۳]. این ابزارها فایل مشکوک را در یک محیط محدودشده که به آن جعبه‌شن[3] گفته می‌شود اجرا کرده و اقدام به ره‌گیری کدها یا توابع سیستمی اجراشده توسط آن‌ها می‌کنند. سپس با بررسی کدها و توابع سیستمی اجراشده توسط فایل مشکوک پی به هدف آن فایل برده و در انتها تعیین می‌کنند که آیا آن فایل بدافزار است یا خیر. امروزه از این‌گونه ابزارها به‌طور گسترده برای شناسایی بدافزار در آزمایشگاه‌های تحلیل بدافزار استفاده می‌شود.

مهم‌ترین مشکل این ابزارها عدم شفافیت است [۴]. یعنی راه‌کارهایی وجود دارد که برنامه تحت تحلیل می‌تواند بفهمد که آیا هم اکنون در درون یک محیط تحلیل اجرا شده است یا خیر. امروزه بدافزارهایی منتشر شده‌اند که از این مشکل موجود در ابزارهای کنونی سوء استفاده کرده و به‌صورت چندشخصیتی عمل می‌کنند. یعنی اگر محیط تحلیل را شناسایی کنند، یک رفتار غیربدخواهانه از خود نشان می‌دهند. در نتیجه ابزار تحلیل نمی‌تواند پی به ماهیت بدخواهانه آن‌ها ببرد [۵]. بنابراین نیاز به تولید محیط تحلیل‌گری می‌باشد که بدافزار به‌راحتی نتواند آن را شناسایی نماید.

هدف ارائه محیط تحلیل‌گری است که تا حد امکان شفاف باشد و راه‌کارهایی که تاکنون برای شناسایی محیط تحلیل‌گر ارائه شده است، روی آن قابل استفاده نباشد. در واقع هدف اصلی این مقاله ارائه راه‌کاری برای جلوگیری از شناسایی محیط تحلیل‌گر می‌باشد. درصورت ایجاد محیط تحلیل‌گری که قابل شناسایی نباشد، یا به‌عبارتی شفاف باشد، بدافزارهای چندشخصیتی به‌سادگی قابل تحلیل خواهند بود. در نتیجه نیاز به تحلیل دستی این‌گونه بدافزارها از بین می‌رود.

همچنین این محیط تحلیل‌گر باید بهینه نیز باشد. یعنی نیاز به منابع سخت‌افزاری زیادی نداشته باشد. چرا که تاکنون چند راه‌کار برای ایجاد یک محیط تحلیل شفاف ارائه شده است، اما هیچ‌کدام از این ابزارها به کاربرد تجاری نرسیده‌اند. دلیل این موضوع عدم بهینگی آن‌ها بوده است [۶]. با توجه به حجم بدافزاری که امروزه منتشر می‌شود استفاده از این‌گونه ابزارها عملاً ممکن نیست، بنابراین نیاز به محیط تحلیلی وجود دارد که در حین شفافیت، بهینه نیز باشد.

۲. پیشینه تحقیق

در زمینه تحلیل بدافزار مبتنی بر روش‌های مجازی‌سازی به‌کمک سخت‌افزار چندین ابزار ارائه شده است.

ابزار HyperSleuth که مبتنی بر HyperDBG ساخته شده است، یکی از ابزارهایی است که برای تحلیل بدافزار با استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار مورد استفاده قرار می‌گیرد [۷]. این ابزار به‌صورت یک فایل اجرایی است، که بر روی رایانه‌ای که ممکن است آلوده به بدافزار باشد، اجرا می‌شود. سپس با استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار اختیار کل سیستم را گرفته و در واقع می‌تواند با سطح دسترسی ring -1 اجرا شود. به این ترتیب ابزار HyperSleuth بر روی سیستم عامل اجرا می‌شود، یعنی دسترسی HyperSleuth از سیستم عامل نیز بیشتر است [۸].

نرم‌افزار Ether یکی دیگر از نرم‌افزارها و چارچوب‌هایی است که برای تحلیل بدافزار ارائه شده است. این نرم‌افزار بر پایه Xen طراحی شده و با استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار توانسته است شفافیت را تا حدی ایجاد نماید [۹].

همچنین ابزار دیگری به‌نام Holography نیز وجود دارد که به‌صورت یک اضافه‌شونده روی QEMU نصب می‌شود و فراخوانی‌های سیستمی اجرا شده توسط پردازش را شنود می‌کند. البته برعکس Ether این ابزار نتوانسته است شفافیت ایجاد نماید، چرا که مبتنی بر QEMU می‌باشد و روش‌هایی برای تشخیص QEMU وجود دارد، که ممکن است توسط بدافزارها مورد استفاده قرار بگیرد. این نکته بسیار حائز اهمیت است [۱۰].

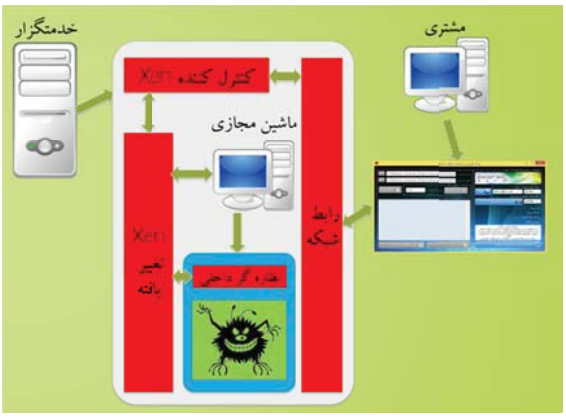
راه‌کاریی برای تحلیل بدافزار مبتنی بر مجازی‌سازی سخت‌افزاری ارائه شده است که تا حدی قابل‌حمل بوده و برای چندین سیستم‌عامل قابل استفاده می‌باشد [۱۱]. همچنین راه‌کارهایی برای شناسایی بدافزار با استفاده از روش‌های مشابه ارائه شده است [۱۲].

[1] Polymorphic
[2] Packer
[3] Sandbox

مشکل مشترک تمامی ابزارهای معرفی‌شده در این بخش، عدم بهینگی می‌باشد. با این‌که می‌توان با استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار محیط تحلیل شفاف ایجاد نمود، اما این کار باعث کاهش شدید سرعت تحلیل بدافزار خواهد شد.

۳. چارچوب پیشنهادی

مؤلفه‌های اصلی این چارچوب در شکل (۱) نشان داده شده است. همان‌طورکه مشاهده می‌شود این چارچوب به‌صورت یک سامانه مشتری[1]/ خدمتگزار[2] طراحی شده است. مشتری از نرم‌افزاری با نام "رابط کاربری سامانه تحلیل بدافزار" استفاده کرده و فایل مشکوک را از طریق پروتکل‌های شبکه به خدمتگزار ارسال می‌نماید. در سمت خدمتگزار مؤلفه‌ای با نام "رابط شبکه" وجود دارد، که وظیفه دریافت و ارسال داده به مشتری را بر عهده دارد.



شکل ۱. نمایی سطح بالا از چارچوب ارائه‌شده در مقاله

خدمتگزار بدافزار دریافت‌شده از مشتری را در نسخه تغییریافته‌ای از ماشین مجازی Xen اجرا می‌کند و اقدام به شنود توابع بومی فراخوانی‌شده توسط پردازش تحت تحلیل می‌نماید. شنود توابع بومی در دو سطح انجام می‌شود. یکی در سطح داخلی و با استفاده از روش تغییر آدرس توابع در جدول SSDT صورت گرفته و دیگری در سطح خارج از ماشین مجازی و به‌کمک فناوری مجازی‌سازی و سخت‌افزار موجود در Xen انجام می‌شود.

در Xen هر ماشین مجازی با یک شماره شناسایی می‌گردد. به این شماره شناسایی، "شماره دامنه" گفته می‌شود. نظاره‌گر ماشین مجازی نیز خودش دارای شماره دامنه صفر است و همیشه با دسترسی بالا یا 0 ring اجرا می‌گردد. مؤلفه "کنترل‌کننده Xen"، وظیفه مدیریت ماشین مجازی و همچنین دریافت لیست توابع بومی فراخوانی‌شده توسط پردازش تحت تحلیل را از مؤلفه "Xen تغییریافته" دارد. خدمتگزار از نسخه ۶۴ بیتی سیستم‌عامل Debian به عنوان سیستم عامل میزبان و از نسخه ۳۲ بیتی و ۶۴ بیتی ویندوز XP به‌عنوان سیستم‌عامل میهمان استفاده می‌کند.

۴. جزئیات پیاده‌سازی چارچوب پیشنهادی

در این بخش ابتدا جزئیات پیاده‌سازی هرکدام از مؤلفه‌های چارچوب پیشنهادی ارائه شده و سپس صحت عملکرد این چارچوب با استفاده از روش‌های ریاضی به اثبات خواهد رسید.

۴-۱. مؤلفه رابط کاربری سامانه تحلیل بدافزار

این مؤلفه توسط مشتری مورد استفاده قرار می‌گیرد. تصویری از این رابط کاربری در شکل (۲) نشان داده شده است.

همان‌طورکه مشاهده می‌شود مشتری می‌تواند با واردکردن آدرس خدمتگزار به آن متصل شده و سپس فایل مشکوک خود را در فیلد "فایل ورودی" انتخاب نماید. همچنین می‌تواند با انتخاب گزینه "فیلتر خروجی" برخی از توابع بومی را از خروجی تحلیل حذف نماید. خروجی این چارچوب به‌صورت XML می‌باشد. این خروجی حاوی لیست فراخوانی‌های انجام‌شده توسط پردازش تحت تحلیل به‌همراه آرگومان‌های هر فراخوانی و مقدار خروجی می‌باشد.

پیاده‌سازی این مؤلفه با استفاده از زبان C# و به‌کمک کتابخانه Telerik صورت گرفته است. اصولاً در سمت مشتری هیچ‌گونه پردازشی صورت نمی‌گیرد. تنها وظیفه این مؤلفه، ارتباط با مؤلفه "رابط شبکه" برای ارسال و دریافت داده می‌باشد. این ارتباط از طریق یک درگاه[3] انتخابی (که پیش‌فرض آن ۲۱۷۰۱ است) و یک پروتکل اختصاصی صورت گرفته است. جزئیات این پروتکل اختصاصی در بخش بعد ارائه خواهد شد.

شکل ۲. رابط کاربری سامانه تحلیل بدافزار

[1] Client
[2] Server
[3] Port

۴-۲. مؤلفه رابط شبکه

همان‌طورکه در شکل (۱) مشاهده می‌شود، مؤلفه رابط شبکه، تنها مؤلفه‌ای است که به‌طور مستقیم با مشتری در ارتباط است. وظیفه این مؤلفه دریافت فایل مشکوک جهت تحلیل از مشتری و همچنین ارسال خروجی تحلیل به مشتری می‌باشد. برای این کار از یک پورت دلخواه شبکه استفاده می‌شود. برای ارسال داده نیز از یک پروتکل اختصاصی استفاده شده است و از پروتکل‌های مشهور انتقال فایل همانند SMB یا FTP استفاده نشده است. برای پیاده‌سازی این مؤلفه از توابع استاندارد شبکه استفاده شده است. هر بسته‌ای که از رابط شبکه به‌سمت مشتری ارسال می‌گردد، به‌صورت زیر است:

<PacketType><PacketSize>:<PacketContents>

<PacketType> نوع بسته را نشان می‌دهد که برای نمونه می‌تواند برابر LOG باشد. <PacketSize> طول بسته و <PacketContents> محتوای بسته را نشان می‌دهد.

به‌عنوان مثال اگر رابط شبکه بخواهد متنی با پیغام xend service stopped به مشتری ارسال نماید، باید بسته‌ای با محتوای زیر به درگاه مدنظر بفرستد.

LOG20:xend service stopped

۴-۳. مؤلفه کنترل‌کننده Xen

این مؤلفه دو وظیفه عمده دارد. وظیفه اول آن مدیریت ماشین مجازی است. یعنی زمانی‌که درخواست تحلیل یک فایل از سمت مشتری ارسال گردد، این مؤلفه اقدام به راه‌اندازی سرویس‌های Xen نموده، سپس یک ماشین مجازی ساخته و فایل دریافتی را در درون آن اجرا می‌نماید. بعد از اتمام زمان تحلیل، اقدام به از کار انداختن سرویس Xen نموده و همچنین نسخه پشتیبانی از دیسک سخت مجازی را بازگردانی می‌کند. در واقع تمامی تغییرات انجام‌گرفته بر روی ماشین مجازی را به حالت اولیه برمی‌گرداند.

۴-۴. مؤلفه Xen تغییریافته

در این چارچوب، از ابزار Xen برای مجازی‌سازی به‌کمک سخت‌افزار استفاده می‌شود. با توجه به این‌که ابزار Xen متن‌باز است، لذا می‌توان یک نسخه شخصی‌سازی‌شده از آن ایجاد کرد. در این چارچوب، تغییراتی در Xen ایجاد شده است که با استفاده از آن می‌توان اقدام به شنود فراخوانی‌های صورت‌گرفته در درون ماشین مجازی Xen نمود.

این تغییرات شامل تغییر مقدار ثبات‌های IA32_SYSENTER_EIP،IA32_SYSENTER_CS و همچنین آدرس وقفه 0x2E در جدول وقفه می‌باشد. این مقادیر برابر صفحاتی قرار می‌گیرند که اصلاً وجود خارجی ندارند. (ما از آدرس 0xEEEEE09D استفاده کرده‌ایم.) لذا هر بار که یک فراخوانی سیستمی رخ می‌دهد، یک خروج از ماشین مجازی به‌خاطر خطای صفحه ایجاد می‌شود، و از آن به بعد می‌توان اطلاعات فراخوانی سیستمی مورد نظر را به‌دست آورد. برای این کار تابع زیر به Xen اضافه شده است:

```
/* tanzime meghdare SYSENTER_CS va SYSENTER_EIP */
inline static void vmx_set_sysenter_msrs(struct domain *d)
{
        u64 new_cs = 0;
        u64 new_eip= 0xEEEEE09D;

        vmx_write_sysenter_msr(GUEST_SYSENTER_CS, new_cs);
        vmx_write_sysenter_msr(GUEST_SYSENTER_EIP, new_eip);

}
```

تابع vmx_write_sysenter_msr دو آرگومان از ورودی می‌گیرد. آرگومان اول آدرس ثباتی است که باید نوشته شود و آرگومان دوم مقدار آن می‌باشد. تابع vmx_write_sysenter_msr به‌صورت زیر نوشته شده است:

```
void vmx_write_sysenter_msr(unsigned long msr, unsigned long value)
{
        if(value != 0)
        {
                __vmwrite(msr, value);
        }
}
```

همان‌طورکه مشاهده می‌شود برای پیاده‌سازی این تابع از تابع داخلی Xen به نام __vmwrite استفاده شده است.

حال که IA32_SYSENTER_EIP و IA32_SYSENTER_CS تغییر داده شده‌اند، چگونه می‌توان خطای صفحه صورت گرفته ناشی از این کار را مدیریت کرد؟ برای این کار باید تابع sh_page_fault تغییر کند. این تابع در هنگام ایجاد خطای صفحه در ماشین مجازی فراخوانی می‌شود. در درون این تابع کدهایی اضافه شده است که با استفاده از آن ابتدا بررسی می‌شود که آیا خطای صفحه رخداده مربوط به فراخوانی سیستمی هست یا خیر. درصورتی‌که مربوط به یک فراخوانی سیستمی باشد، ابتدا تابع analysis_handle_syscall را فراخوانی کرده، تا کارهای لازم برای نمایش خروجی انجام شود و تابع __vmwrite برای نوشتن مقدار اصلی IA32_SYSENTER_EIP در ثبات RIP فراخوانی می‌گردد.

با انجام این تغییرات در Xen، همان‌طورکه در شکل (۳) مشاهده می‌شود به‌ازای هر فراخوانی سیستمی ماشین مجازی متوقف شده، تحلیل‌های لازم (شامل نمایش نام فراخوانی سیستمی به همراه لیست آرگومان‌ها و همچنین اعمال برخی محدودیت‌ها) انجام شده و در انتها فرآیند اجرای ماشین مجازی ادامه می‌یابد.

شکل ۳. روال اجرای هر فراخوانی سیستمی در ماشین مجازی

۴-۵. مؤلفه نظاره‌گر داخلی

همان‌طورکه گفته شد روش شنود با استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار با این‌که روشی شفاف می‌باشد، اما نسبت به سایر روش‌های تحلیل دارای میزان بهینگی کم‌تری است. راه حل این مشکل استفاده از یک ساختار لایه‌ای است. یعنی یک لایه از عملیات شنود در داخل ماشین مجازی و به‌وسیله مؤلفه "نظاره‌گر داخلی" انجام می‌گیرد. در ابتدا تنها مؤلفه "نظاره‌گر داخلی" اقدام به شنود می‌نماید و از فناوری مجازی‌سازی به‌کمک سخت‌افزار برای شنود استفاده نمی‌شود. تا زمانی‌که این خطر حس شود که نظاره‌گر داخلی ممکن است شناسایی گردد، یا این‌که توان شنود بخشی از کدها را نداشته باشد، در این لحظه شنود در لایه داخلی کاملاً متوقف شده و به لایه پایین‌تر انتقال می‌یابد. اضافه‌کردن مؤلفه "نظاره‌گر داخلی" ممکن است باعث عدم شفافیت چارچوب گردد. لذا باید از روش‌هایی برای شنود در این مؤلفه استفاده شود، که باعث از بین‌رفتن شفافیت چارچوب نشود.

یک چارچوب تحلیل بدافزار تک‌لایه‌ای با نام L_A را درنظر بگیرید. فرض کنید برای شناسایی این چارچوب n روش مختلف وجود دارد. برای استفاده از هرکدام از این روش‌ها باید m دستور اسمبلی(که با i نشان داده می‌شود) فراخوانی گردد. این روش‌ها را از I_1 تا I_n نام‌گذاری می‌نماییم. مجموعه روش‌هایی که برای شناسایی چارچوب L_A می‌تواند مورد استفاده قرار گیرد، را I_A می‌نامیم. در نتیجه داریم:

$$I_A = \{I_1 , I_2 , I_3 , \dots , I_n\} \qquad (۱)$$

$$\begin{cases} I_1 = \{i_{1,0} , i_{1,1} , i_{1,2} , \dots , i_{1,m1}\} \\ I_2 = \{i_{2,0} , i_{2,1} , i_{2,2} , \dots , i_{2,m2}\} \\ \quad\quad\quad . \\ \quad\quad\quad . \\ \quad\quad\quad . \\ I_n = \{i_{n,0} , i_{n,1} , i_{n,2} , \dots , i_{n,mn}\} \end{cases} \qquad (۲)$$

واضح است برای اینکه یک چارچوب تک‌لایه‌ای شفاف باشد، باید $I_A = \emptyset$ باشد. اما حال یک چارچوب دولایه‌ای را با نام‌های L_A و L_B را درنظر بگیرید. لایه سطح بالا L_A و لایه سطح پایین L_B می‌باشد. برای اینکه کل چارچوب شفاف باشد، لازم است پایین‌ترین لایه شفاف باشد. یعنی $I_B = \emptyset$. از طرفی طبق معادله ۱، فرض کنید n روش شناسایی برای لایه L_A وجود داشته باشد. همچنین فرض کنید k دنباله دستور مختلف وجود دارد که لایه L_A قادر به شنود آن‌ها نیست. اگر مجموعه تمامی دنباله دستورات غیرقابل شنود در لایه L_A را برابر I_{AU} درنظر بگیریم، آنگاه داریم:

$$I_{AU} = \{I_1 , I_2 , I_3 , \dots , I_k\} \qquad (۳)$$

همان‌طورکه گفته شد در یک ساختار لایه‌ای، در یک لحظه تنها یک لایه در حال شنود فایل مشکوک می‌باشد. به محض این‌که لایه در حال شنود احساس کند که نمی‌تواند عملیات شنود را به‌گونه‌ای شفاف ادامه دهد، عملیات را به لایه پایین‌تر منتقل می‌کند. فرض کنید مجموعه دنباله دستوراتی که باعث می‌شود عمل شنود از لایه L_A به لایه L_B منتقل شود را T_{AB} بنامیم. و همچنین تابع $D_A(x)$ را به‌صورت زیر تعریف کنیم:

$$D_A(x) = \begin{cases} 1, & \text{دستورات } x \text{ قبل از فراخوانی در } L_A \text{ قابل شناسایی نباشند} \\ 0, & \text{دستورات } x \text{ قبل از فراخوانی در } L_A \text{ قابل شناسایی باشند} \end{cases} \qquad (۴)$$

آنگاه یک چارچوب دولایه‌ای کاملاً شفاف است، اگر و تنها اگر سه شرط زیر برقرار باشد:

$$\begin{cases} I_B = \varnothing \\ I_A \cup I_{AU} \subseteq T_{AB} \\ \forall x \in T_{AB} , D_A(x) = 1 \end{cases} \qquad (۵)$$

اثبات: برای اثبات این‌که این سه شرط، شرط لازم شفافیت هستند، می‌توان از برهان خلف استفاده کرد. فرض کنید شرط اول برقرار نباشد، آنگاه واضح است که وقتی پایین‌ترین سطح تحلیل شفاف نیست، در نتیجه کل چارچوب نیز نمی‌تواند شفاف باشد. اگر شرط دوم برقرار نباشد، آنگاه: $\exists x \in (I_A \cup I_{AU}), x \notin T_{AB}$ و در نتیجه دنباله دستوراتی وجود دارد که توان شناسایی لایه L_A را داشته، ولی باعث انتقال عملیات شنود به لایه پایین‌تر نمی‌گردند، در نتیجه x می‌تواند وجود محیط

تحلیل را شناسایی نماید. اگر شرط سوم برقرار نباشد، یعنی دنباله دستوراتی وجود دارد که باعث انتقال عملیات شنود به لایه پایین‌تر می‌شود، اما توسط لایه L_A قابل شناسایی نیست. اگر قابل شناسایی نباشد، در نتیجه نمی‌توان آن را به لایه پایین‌تر انتقال داد، که این با فرض برهان در تناقض است. برای اثبات کافی بودن این سه شرط داریم:

$$I_A \cup I_{AU} \subseteq T_{AB} \rightarrow \forall x \in (I_A \cup I_{AU}) , D_A(x) = 1$$

و این یعنی هر دنباله دستوری که برای لایه L_A مشکل ایجاد می‌کند، قبل از فراخوانی کامل توسط این لایه قابل شناسایی است. در نتیجه عملیات به لایه پایین‌تر منتقل می‌شود، و با توجه به این‌که در لایه پایین‌تر داریم $I_B = \emptyset$، لذا سامانه شفاف است. قضیه بالا را می‌توان به n لایه نیز تعمیم داد.

حال باید مؤلفه "نظاره‌گر داخلی" به‌گونه‌ای طراحی شود که دو شرط موجود در معادله (۵) را داشته باشد. در بخش بعدی طریقه پیاده‌سازی این مؤلفه ارائه خواهد شد.

۴-۶. طریقه پیاده‌سازی مؤلفه نظاره‌گر داخلی

مؤلفه نظاره‌گر داخلی در سطح هسته عمل کرده و از روش "تغییر آدرس توابع در جدول SSDT" استفاده می‌کند. دلیل انتخاب این روش در بخش بعدی ارائه می‌گردد. ساختمان داده جدول SSDT در زیر ارائه شده است:

```
struct SysServiceTable{
        void **ServiceTableArray;
        unsigned long CounterTableindex;
        unsigned long ServiceLimitNumber;
        void** APIArguments;
};
```

فیلد ServiceTableArray به یک آرایه حاوی آدرس تمام توابع سیستمی سرویس‌دهنده در جدول SSDT اشاره می‌کند. بنابراین iامین عنصر آرایه ServiceTableArray حاوی آدرس تابع سیستمی شماره i است [۱۳]. در نتیجه برای تغییر آدرس توابع سیستمی کافیست مقدار اندیس‌های این آرایه تغییر داده شود.

این ساختمان داده در هسته سیستم عامل ویندوز که فایل ntoskrnl.exe است، قرار دارد. همان‌طورکه می‌دانید در سیستم‌عامل‌های ۳۲ بیتی هر پردازش به ۴ گیگابایت حافظه اصلی دسترسی دارد، که دو گیگابایت آن مربوط به سطح کاربر و ۲ گیگابایت دیگر آن مربوط به سطح هسته است [۱۴]. پردازش‌ها به‌طور مستقیم نمی‌توانند داده‌های ۲ گیگابایت سطح هسته را بخوانند، مگر با دو روش زیر:

- خودشان یک درایور در سیستم بارگذاری کرده و آن درایور حافظه سطح هسته را بخواند.
- از روش‌های نسخه‌برداری استفاده نمایند. یعنی از کل حافظه نسخه‌برداری کرده و محل جدول SSDT را با استفاده از برخی نشانه‌ها کشف کرده، سپس اقدام به خواندن محتوای SSDT نمایند. اصولاً دو روش برای نسخه‌برداری از حافظه اصلی در سطح کاربر وجود دارد، یکی خواندن مستقیم حافظه اصلی با استفاده از آدرس شی \device\physicalmemory به‌کمک تابع CreateFile می‌باشد و دیگری استفاده از تابع بومی NtSystemDebugControl برای نسخه‌برداری از حافظه اصلی است [۱۵].

با توجه به این‌که محتوای ساختمان داده SSDT به‌طور کامل در فایل ntoskrnl.exe قرار دارد، در نتیجه پردازش سطح کاربر تنها با استفاده از دو روش بالا توان شناسایی تغییرات صورت‌گرفته بر روی SSDT را دارد. بنابراین مؤلفه "نظاره‌گر داخلی" به‌محض این‌که یکی از دو رفتار بالا را مشاهده می‌نماید، اقدام به انتقال عملیات شنود به لایه پایین‌تر می‌کند. در لایه پایین‌تر از روش مجازی‌سازی به‌کمک سخت‌افزار استفاده خواهد شد، که روشی شفاف برای تحلیل بدافزار است.

ذکر این نکته مهم است که برای تغییر جدول SSDT از یک درایور استفاده شده است. بنابراین یکی از راه‌های شناسایی این چارچوب می‌تواند شناسایی درایور نصب‌شده بر روی سیستم باشد. برای جلوگیری از شناسایی چارچوب با استفاده از این روش به‌صورت زیر عمل می‌شود:

۱. ابتدا درایوری در سیستم نصب شده و بارگذاری می‌گردد.
۲. این درایور آدرس توابع سیستمی موجود در جدول SSDT را تغییر می‌دهد.
۳. درایور از سیستم حذف شده و کلیه آثار آن درایور همانند کلیدهای رجیستری و فایل آن از سیستم به‌گونه‌ای غیرقابل بازگشت حذف می‌شود.
۴. فایل مشکوک در سیستم اجرا می‌گردد.

به‌عبارتی در زمانی‌که فایل مشکوک در سیستم اجرا می‌شود، هیچ اثری از درایور تغییردهنده جدول SSDT در سیستم وجود

ندارد. ذکر این نکته مهم است که آدرس جدید توابع قلاب[1] در یک صفحه جدید از حافظه اصلی قرار گرفته‌اند. به این ترتیب حتی بعد از حذف درایور نیز این توابع در حافظه وجود دارند. برای ایجاد یک صفحه جدید در حافظه سطح هسته از کدهایی همانند زیر استفاده شده است:

```
void *kSpaceAddr;

PMDL mem_desc = IoAllocateMdl((PVOID)&kSpaceAddr, 500, FALSE, FALSE, NULL);

MmProbeAndLockPages(mem_desc, KernelMode, IoReadAccess + IoWriteAccess);

RtlZeroMemory(mem_desc->StartVa, 500);
```

از تابع IoAllocateMdl برای ساخت یک "لیست توصیفگر حافظه" استفاده شده است. سپس این لیست توصیفگر قفل شده و دسترسی خواندن و نوشتن به آن داده شد. به این ترتیب از سیستم ۵۰۰ بایت حافظه اصلی سطح هسته گرفته شده است. از این ۵۰۰ بایت می‌توان برای کپی‌کردن کدهای توابع قلاب استفاده کرد.

با توجه به این‌که کدها به جایی خارج از درایور منتقل می‌شوند، باید برخی مسائل همانند هم‌ترازی حافظه و آدرس توابع هسته مد نظر قرار گیرد. برای به‌دست آوردن آدرس توابع موجود در فایل‌های ntoskrnl.exe و hal.dll از تابع MmGetSystemRoutineAddress استفاده شده است.

همچنین یکی دیگر از راه‌کارهایی که ممکن است باعث شناسایی محیط تحلیل شود، نام و محل قرارگیری فایل مشکوک در درون ماشین مجازی است. برای جلوگیری از شناسایی با استفاده از این روش، هر بار محل قرارگیری و نام فایل تغییر می‌کند.

۴-۷. اثبات شفاف‌بودن چارچوب ارائه‌شده

همان‌طور که گفته شد، این چارچوب در دو لایه اقدام به تحلیل می‌کند. لایه سطح پایین با استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار انجام می‌گیرد و آن را L_B می‌نامیم. لایه سطح بالا نیز توسط مؤلفه "نظاره‌گر داخلی" و با استفاده از روش قلاب‌اندازی به جدول SSDT صورت می‌گیرد. ما این لایه را L_A می‌نامیم. همان‌طورکه در بخش‌های قبلی اشاره شد، داریم: $I_B = \emptyset$. در بخش‌های قبلی گفته شد که دو روش برای شناسایی لایه L_A وجود دارد. بنابراین مجموعه I_A دارای دو عضو خواهد بود:

$$I_A = \{File\_PhyMemory, NtSystemDebugControl\} \quad (۶)$$

عضو File_PhyMemory اشاره به خواندن مستقیم حافظه اصلی با استفاده از آدرس شی \device\physicalmemory به‌کمک تابع CreateFile دارد، و عضو NtSystemDebugControl به روش نسخه‌برداری از حافظه اصلی با استفاده از تابعی به همین نام اشاره می‌کند.

لایه L_A توان تحلیل کدهای سطح هسته را ندارد. برای تحلیل این کدها نیاز به استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار و شنود دنباله کدهای اسمبلی برخی روال‌های کنترل "بسته درخواست ورودی/خروجی[2]" می‌باشد. بنابراین I_{AU} به‌صورت زیر تعریف می‌گردد:

$$I_{AU} = \{Custom\_Kernel\_Code\} \quad (۷)$$

برای این‌که این چارچوب شفاف باشد، باید سه شرط اشاره‌شده در معادله (۵) برقرار باشد. برقراری شرط اول در بخش‌های قبل مشخص شد. جهت برقراری شرط دوم نیز مجموعه T_{AB} باید حداقل دارای اعضای زیر باشد:

$$\{File\_PhyMemory, NtSystemDebugControl, Custom\_Kernel\_Code\} \subseteq T_{AB} \quad (۸)$$

شرط سوم نیز می‌گوید که قبل از اجرای سه دنباله کد موجود در معادله بالا، باید روشی برای شناسایی این کدها توسط مؤلفه "نظاره‌گر داخلی" وجود داشته باشد. لایه L_A به سادگی قادر به شناسایی دو دنباله کد File_PhyMemory و NtSystemDebugControl قبل از فراخوانی کامل آن هست. برای شناسایی این دو دنباله کد کافیست توابع NtSystemDebugControl و CreateFile را در جدول SSDT قلاب‌اندازی نموده و با بررسی آرگومان‌ها به‌سادگی می‌توان این دو دنباله را شناسایی نمود. در نتیجه:

$$\begin{cases} D_A(File\_PhyMemory) = 1 \\ D_A(NtSystemDebugControl) = 1 \end{cases} \quad (۹)$$

پردازش تحت تحلیل برای اجرای کد در سطح هسته نیاز به بارگذاری درایور دارد. برای بارگذاری درایور در سطح پایین دو راه وجود دارد. یکی استفاده از تابع NtLoadDriver که راه اصلی این کار است و دیگری استفاده از تابع NtSetSystemInformation می‌باشد. بنابراین با قلاب‌اندازی به این دو تابع در جدول SSDT نیز می‌توان بارگذاری هرگونه

[1] Hook

[2] I/O Request Packet

درایور در سیستم را شناسایی نمود. لذا داریم:

$$D_A(\text{Custom\_Kernel\_Code})=1 \rightarrow (\forall x \in T_{AB},\ D_A(x)=1)$$

بنابراین شرط سوم برای این چارچوب برقرار است. جهت برقراری شرط دوم هر بار که یکی از سه دنباله کد موجود در معادله ۸ اجرا شود، باید عمل انتقال شنود به لایه پایین‌تر صورت گیرد. که این کار در چارچوب پیشنهادی انجام می‌گیرد.

لذا شرط دوم نیز برقرار است.

۵. ارزیابی چارچوب پیشنهادی

ارزیابی این چارچوب در دو مرحله صورت می‌گیرد. در مرحله اول شفافیت و در مرحله دوم میزان بهینگی سامانه مورد بررسی قرار می‌گیرد.

جدول ۱. نتایج آزمون بهینگی سامانه

Env/Packer	Sandboxie	Avast	Comodo	VMWare	VirtualBox	Ether	Parsa	Our Framework
Enigma	×	√	√	×	×	√	√	√
ExeCryptor	×	√	√	√	√	√	×	√
NoobyProtect	×	√	√	×	√	√	√	√
Obsidium	×	√	√	×	×	√	√	√
Themida	√	√	√	×	×	×	√	√
VMProtect	×	√	√	×	√	√	√	√
Aegis	×	×	×	×	×	√	√	√

۵-۱. ارزیابی شفافیت چارچوب پیشنهادی

برای بررسی میزان شفافیت محیط‌های تحلیل یا ماشین‌های مجازی، ابتدا یک برنامه ساده نوشته و سپس آن را با استفاده از فشرده‌سازهای دارای قابلیت شناسایی محیط تحلیل و "رمزگذارهای کاملاً غیرقابل شناسایی[1]" محافظت کرده و در درون محیط‌های تحلیل اجرا کرده‌ایم. اگر برنامه نوشته‌شده دقیقاً به همان صورت طراحی‌شده اجرا گردد، یعنی محیط نسبت به آن رمزگذار یا فشرده‌ساز شفاف است. خروجی آزمون صورت‌گرفته در جدول (۱) نشان داده شده است.

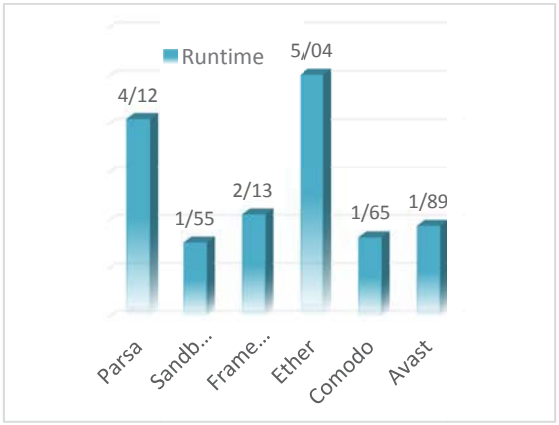
همان‌طورکه مشاهده می‌شود چارچوب ارائه‌شده در این مقاله تنها محیطی است که توسط هیچ‌کدام از فشرده‌سازهای مشهور امروزی که از ده‌ها روش همانند قرص قرمز[2]، خطاهای پردازنده یا روش‌های زمانی برای شناسایی ماشین مجازی و محیط تحلیل استفاده می‌کنند، قابل شناسایی نیست. حتی چارچوب Ether که در [۹] گفته شده کاملاً شفاف است، توسط قابلیت Anti-Trace فشرده‌ساز ExeCryptor شناسایی شده است. این قابلیت با استفاده از توابع QueryPerformanceCounter از روش‌های زمانی برای شناسایی محیط تحلیل استفاده می‌نماید. دلیل این موضوع وجود برخی ایرادات در پیاده‌سازی چارچوب Ether می‌باشد [۱۶].

۵-۲. ارزیابی بهینگی چارچوب پیشنهادی

برای ارزیابی میزان بهینگی چارچوب پیشنهادی ۳۰۰ بدافزار از منابع مختلف اینترنتی همچون VirusSign، VirusShare و vxheaven دریافت شد. سپس این بدافزارها به محیط‌های تحلیل مختلف داده شده و سرعت رسیدن بدافزار به یک نقطه خاص از کد با استفاده از نرم‌افزار API Monitor و همچنین Process Monitor اندازه‌گیری شد. نتایج آزمون بهینگی برای بدافزارهای سطح کاربر در شکل (۴) نشان داده شده است.

همان‌طورکه در این شکل مشاهده می‌شود چارچوب پیاده‌سازی‌شده در این مقاله(که با نام Framework نشان داده شده است) با این‌که یک چارچوب شفاف است، اما تفاوت بسیار کمی از لحاظ میانگین زمان اجرا نسبت به چارچوب غیرشفافی همچون Sandboxie دارد. همچنین مشاهده می‌شود که به‌خاطر ایجاد شفافیت در محیط تحلیل Ether این محیط با کندی زیادی نسبت به سایر چارچوب‌ها اقدام به تحلیل بدافزار می‌نماید. این کندی به حدی است که این محیط را برای تحلیل بدافزار در دنیای امروز ناکارآمد کرده است.

[1] Fully Undetectable Crypter (FUD)
[2] Red Pill

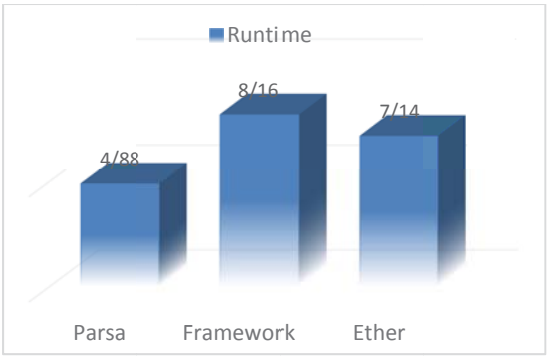


شکل ۴. زمان اجرای محیط‌های تحلیل برای بدافزارهای سطح کاربر

نتایج آزمون بهینگی برای بدافزارهای سطح هسته نیز در شکل (۵) نشان داده شده است.

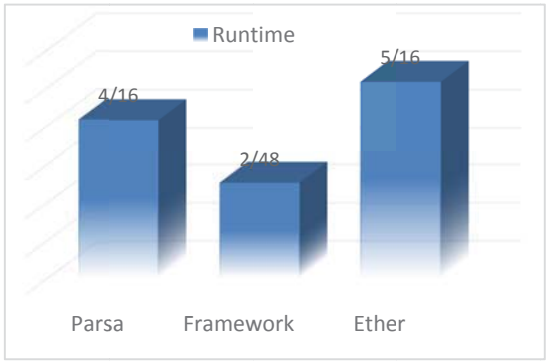
نکته: برای انجام این آزمون از یک رایانه همراه HP dv6-7070se دارای پردازنده Intel Core i7-2670QM به‌همراه ۶ گیگابایت حافظه اصلی استفاده شده است.

نکته: جعبه‌شن‌های Avast، Comodo و SandBoxie از بدافزارهای سطح هسته پشتیبانی نمی‌کنند، در نتیجه در آزمون بهینگی برای بدافزارهای سطح هسته شرکت داده نشده‌اند.



شکل ۵. زمان اجرای محیط‌های تحلیل برای بدافزارهای سطح هسته

همان‌طورکه مشاهده می‌شود زمان اجرای چارچوب ارائه‌شده در مقاله برای بدافزار سطح هسته از سایر محیط‌های تحلیل بیش‌تر است. در نتیجه نسبت به آن‌ها سربار بیش‌تری دارد. با اجرای بدافزارهای موجود در مجموعه آزمون و بررسی لیست درایورهای سیستم به این نتیجه رسیدیم که تنها ۵/۸ درصد بدافزارها حاوی درایور سطح هسته هستند، یا این‌که سعی در تغییر مستقیم سکتورهای دیسک سخت در جهت تغییر فایل‌های هسته می‌نمایند. بنابراین اگر این ضریب در خروجی موجود در شکل (۴و۵) درنظر گرفته شود، آنگاه شکل (۶) حاصل می‌شود. همان‌طورکه می‌بینید میانگین کل زمان اجرای محیط تحلیل ارائه‌شده در این مقاله نسبت به سایر محیط‌های تحلیل کم‌تر می‌باشد و این نشان از بهینه‌بودن این سامانه برای تحلیل خودکار بدافزار دارد. همچنین از طرفی جعبه‌شن‌های Avast، Comodo و Sandboxie اصولاً برای محدودسازی به‌کار می‌روند، بنابراین دارای عمق تحلیل کم‌تری هستند. با توجه به اینکه میزان سربار چارچوب ارائه‌شده نسبت به این جعبه‌شن‌ها بسیار کم است، می‌توان گفت که این میزان سربار نیز به‌خاطر تفاوت عمق تحلیل چارچوب ارائه‌شده با جعبه‌شن‌های مذکور می‌باشد. بنابراین می‌توان نتیجه گرفت این سامانه به حدی بهینه است که توان تحلیل میزان بدافزارهای منتشرشده در روز را با توجه به آمارهای منتشرشده برای سال‌های ۲۰۱۳ و ۲۰۱۴ دارد.



شکل ۶. زمان اجرای محیط‌های تحلیل برای بدافزارهای سطح کاربر

۷. نتیجه گیری

تاکنون ابزارهای تحلیل بدافزاری که ارائه شده بود یا دارای مشکل شفافیت بوده‌اند یا اینکه مشکل بهینگی داشته‌اند. در این مقاله چارچوب تحلیل بدافزاری ارائه شد که شفاف و بهینه است. برای رسیدن به شفافیت همانند کارهای قبلی از فناوری مجازی‌سازی به‌کمک سخت‌افزار استفاده شد. اما برای حل مشکل بهینگی یک ساختار لایه‌ای ایجاد شد تا حجم عظیمی از بدافزارهای منتشرشده در روز بدون نیاز به استفاده شدید از فناوری مجازی‌سازی به‌کمک سخت‌افزار مورد تحلیل قرار گیرند. بنابراین این چارچوب علاوه‌بر این‌که در برابر بدافزارهای چندشخصیتی مقاوم است، توان تحلیل میزان بدافزار منتشرشده در روز را بدون نیاز به اضافه‌کردن منابع سخت‌افزاری جدید دارد.

این چارچوب به‌صورت مشتری/خدمتگزار عمل می‌کند. مشتری فایل را به خدمتگزار ارسال می‌دارد. خدمتگزار فایل اجرایی مشکوک را در داخل ماشین‌مجازی اجرا کرده و فعالیت‌های انجام‌گرفته توسط آن را به مشتری ارائه می‌دهد. برای تحلیل از دو لایه استفاده می‌شود. لایه پایین‌تر از فناوری

مجازی‌سازی به‌کمک سخت‌افزار استفاده می‌کند. این فناوری با این‌که شفاف است، اما بسیار کند عمل می‌کند. برای این‌که این مشکل حل شود از یک لایه داخلی مبتنی بر روش قلاب‌اندازی به جدول SSDT استفاده می‌شود. این لایه با این‌که شفاف نیست، اما طوری طراحی شده است که قبل از این‌که توسط بدافزار کشف شود، کلیه نشانه‌هایش را از سیستم حذف کرده و عملیات تحلیل را به لایه پایین‌تر واگذار می‌کند.

با آزمون‌های صورت‌گرفته مشخص شد که این سامانه با این‌که برعکس سامانه‌های قبلی مشکل عدم بهینگی ندارد، همچنان شفاف نیز می‌باشد. بنابراین تنها چارچوبی است که توان تحلیل این حجم از بدافزار را با استفاده از فناوری مجازی‌سازی به‌کمک سخت‌افزار دارد.

به‌عنوان کارهای آتی می‌توان محیط مشابهی برای سیستم‌عامل‌های تلفن همراه مخصوصاً Android ارائه داد. همچنین می‌توان به چارچوب ارائه‌شده امکان بررسی قانونی[1] اضافه کرد. به‌این‌ترتیب روت‌کیت‌های سطح هسته با بررسی تغییرات صورت گرفته بر نقاط حساس هسته سیستم‌عامل قابل شناسایی خواهند بود.

۸. مراجع

[1] Presti, K. “McAfee Sees Biggest Malware Increase In Four Years”; 4 September 2010. [Online]. Available: http://www.crn.com/news/security/240006717/mcafeesees-biggest-malware-increase-in-four-years.htm?itc=refresh.

[2] “Number of the Year: Kaspersky Lab is Detecting 315,000 New Malicious Files Every Day”; 10 December 2013. [Online].Available:http://www.kaspersky.com/about/news/virus/2013/number-of-the-year.

[3] Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. “A Survey on Automated Dynamic Malware”; ACM Computing Surveys, 2012, 44(2), 1-42.

[4] Deng, Z.; Xu, D.; Zhang, X.; Jiang, X. “IntroLib: Efficient and Transparent Library Call Introspection for Malware Forensics”; in The Proceedings of the Twelfth Annual DFRWS Conference, 2012.

[5] Balzarotti, D.; Cova, M.; Karlberger, C.; Kruegel, C.; Kirda, E.; Vigna, G. “Efficient Detection of Split Personalities in Malware”; in The 17th Annual Network & Distributed System Security Conference, San Diego, 2010.

[6] Fattori, A.; Paleari, R.; Martignoni, L.; Monga, M. “Dynamic and Transparent Analysis of Commodity”; in 25th IEEE/ACM International Conference on Automated, Antwerp. 2010.

[7] "Live and Trustworthy Forensic Analysis of Commodity Production Systems," in 13th International Symposium on Recent Advances in Intrusion Detection, Ottawa, 2010.

[8] Dinaburg, A.; Royal, P.; Sharif, M.; Lee, W. “Ether: Malware Analysis via Hardware Virtualization Extensions”; in 15th ACM Conference on Computer and Communications Security, 2008.

[9] Dai, S.-Y.; Fyodor, Y.; Wu, J.-S.; Lin, C.-H.; Huang, Y.; Kuo, S.-Y. “Holography: A Hardware Virtualization Tool for Malware Analysis”; in 15th IEEE Pacific Rim International Symposium on Dependable Computing, Shanghai, 2009.

[10] Pfoh, J.; Schneider, C.; Eckert, C. “Nitro: Hardware-Based System Call Tracing for Virtual Machines”; in 6th International Workshop on Advances in Information and Computer Security, Tokyo, 2011.

[11] Yan, L.-K.; Jayachandra, M.; Zhang, M.; Yin, H. “V2E: Combining Hardware Virtualization and Software Emulation for Transparent and Extensible Malware Analysis”; in VEE'12 Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments. 2012.

[12] Schreiber, S. “Undocumented Windows 2000 secrets a programmers cookbook”; Addison-Wesley. 2001.

[13] Russinovich, M.; Solomon, D.; Ionescu, A. “Windows Internals”; Part 1, Microsoft Press, 2012.

[14] Zhang, J.; Liu, S.; Peng, J.; Guan, A. “Techniques of user-mode detecting System Service Descriptor Table”; in Proceedings of the 13th International Conference on Computer Supported Cooperative Work in Design. 2009.

[15] Pék, G.; Bencsáth, B.; Buttyán, L. “nEther: In-guest Detection of Out-of-the-guest Malware Analyzers”; in Proceedings of the Fourth European Workshop on System Security. 2011.

[16] Funk, C.; Garnaeva, M. “Kaspersky Security Bulletin 2013. Overall Statistics for 2013”; Kaspersky Lab, 10 December 2013. http://securelist.com/analysis/kaspersky-security-bulletin/58265/kaspersky-security-bulletin-2013-overall-statistics-for-2013/ [Accessed 2014 December 01].

[1] Forensics