# Ein Überblick über den Artikel **Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis**

## Erstellt im Rahmen des Seminars aus Programmiersprachen

Heinz Deinhart (9326241) <heinz@tilab.tuwien.ac.at>

Wintersemester 2007

**Zusammenfassung**

Dieser Artikel entstand im Rahmen eines Seminars. Er fasst einen Artikel über Panorama zusammen und vergleicht ihn mit anderen Arbeiten. Panorama ist ein systemweites, feingekörntes und auf dynamischem Tainting aufgebautes Framework, welches, ohne auf Heuristiken oder Signaturen zurückgreifen zu müssen, das Verhalten von potentiellem Schadcode analysieren kann.

# 1 Einführung

Im Rahmen des Seminars aus Programmiersprachen wurde zunächst ein Artikel aus einer vorgegebenen Liste ausgewählt und den anderen Teilnehmern präsentiert. Der Autor hat seinen Vortrag über den Artikel *Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis* [VNJ+07] gehalten. Danach galt es, einen weiterführenden Artikel aus demselben Themenbereich auszuwählen. Die Suche gestaltete sich gleichzeitig schwierig wie einfach, da einerseits der Artikel noch sehr neu war, aber andererseits einige Mitautoren vor kurzem einen geeigneten Artikel veröffentlicht hatten.

In dem ersten Artikel wird eine Implementierung eines Client-basierten Cross-site Scripting (XSS) Abwehrmechanismus erläutert, der auf einer Kombination aus dynamischem und statischem Tainting basiert. XSS ist eine Attacke, die speziell auf Web Applikationen ausgerichtet ist und dazu dient, sensible Daten von Benutzern dieser Applikationen für Dritte verfügbar zu machen. Die Implementierung versucht, diesem Problem mit Tainting von Java Script Code Herr zu werden. Tainting wird in dieser Zusammenfassung noch erläutert werden. Das besondere an dieser Implementierung ist der Ansatz, das Problem auf dem Client zu lösen. Das ermöglicht dem geneigten Benutzer, selbst für Sicherheit zu sorgen, falls er z.B. gezwungen ist, schlecht gewartete bzw. gesicherte Web Applikationen zu benutzen.

Für die Suche nach einem geeigneten Artikel hat sich der Autor zum Ziel gesetzt, einen Artikel zu finden, der wieder einen Ansatz auf der Client Seite beinhaltet und dies ebenfalls mittels Tainting versucht. Es war kein Artikel zu finden, der sich im Speziellen mit XSS befasst. Der Autor hat sich deshalb für den Artikel *Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis* [YSE+07] entschieden, der einen verallgemeinerten Ansatz verfolgt, und zwar beliebige Angriffe auf sensible Daten zu erkennen und zu analysieren.

Das folgende Kapitel dieses Artikels beinhaltet eine Zusammenfassung des Panorama Artikels mit besonderem Fokus auf die grundsätzliche Architektur und das Dynamische Tainting. Im Anschluss werden kurz die Hauptunterschiede zu dem XSS Artikel und anderen Arbeiten zusammengefasst.

# 2 Zusammenfassung des Panorama Artikels

## 2.1 Einführung

Bösartige Schadprogramme (Malware) stellen ein ernstzunehmendes und komplexes Problem dar. Einerseits erzeugen solche Programme Millionen Dollar an Schaden. Andererseits veröffentlichen sogar renommierte Software-Häuser Programme, die als Malware klassifiziert werden können. Ein Beispiel dafür sind Google Desktop oder der Sony Media Player, welcher ein Rootkit installiert.

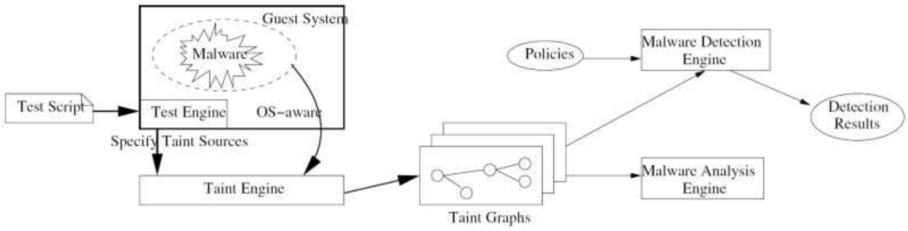
Das Erkennen von Malware wird durch die Anzahl der möglichen Techniken dieser Programme enorm erschwert. Heutzutage wird Malware zumeist mit Signaturen oder Heuristiken erkannt. Programme, welche mit Signaturen arbeiten, können niemals neue Ansätze von Schadcode erkennen. Heuristiken können das zwar theoretisch, allerdings ist es praktisch nicht möglich, wirklich zutreffende Heuristiken zu entwickeln, da es keine allgemein gültigen Unterscheidungskriterien zwischen böswilliger und gutartiger Software gibt. Mit Heuristiken können Programme anhand von verschiedenen Merkmalen wie z.B. verwendeten System Calls klassifiziert werden, welche aber oft von gutartiger Systemsoftware ebenfalls verwendet werden. Autoren von Malware sind natürlich bemüht, das Programm so unauffällig wie irgend möglich zu gestalten.

Panorama stellt einen neuen Ansatz dar, der die tatsächlichen Auswirkungen von potentiellem Schadcode untersucht. Dies findet in einem dreistufigen Prozess statt, der aus Test, Überwachung und Analyse besteht. Ein potentiell bösartiges Programm wird dabei in eine speziell abgeschottete Analyseumgebung geladen. Dann werden Benutzereingaben simuliert und die Reaktion des Programms darauf wird überwacht. Dabei werden alle verdächtigen Verhaltensweisen aufgezeichnet. Diese Aufzeichnungen können im Anschluss an die Tests für eine Analyse verwendet werden.

Die vorgestellte Variante von Panorama wurde auf das Betriebssystem Windows XP ausgelegt. Andere Betriebssysteme lassen sich auch verwenden, nachdem diverse Anpassungen dafür portiert wurden.

## 2.2 Architektur

Die Architektur von Panorama besteht aus folgenden Komponenten: Taint Engine, Test Engine, Malware Detection Engine und Malware Analysis Engine.




Die Aufgabe der Test Engine ist es, Tests automatisiert ablaufen zu lassen, die durch die Test Scripts spezifiziert werden. Die Tests beinhalten die Generierung von sensiblen Daten, die aber nicht für das zu testende Programm bestimmt sind. Dieses Programm wird nun auf potentiellen Zugriff und Weiterverarbeitung der sensiblen Daten hin überwacht. Dazu werden die sensiblen Daten markiert (Tainting) und die Taint Engine kümmert sich um das Propagieren dieser Markierungen. Dies erfolgt systemweit und "fine-grained", also sehr feinkörnig. Diese Taint Engine erzeugt Taint Graphs, welche den Weg von markierten Daten von deren Erzeugung an durch das ganze System aufzeichnen und dadurch nachvollziehbar machen. Diese Graphen können mit Hilfe der Malware Analysis Engine von Experten manuell begutachtet werden. Die Malware Detection Engine ermöglicht es, mit Hilfe eines Sets aus Policies mögliche Malware automatisch zu erkennen. Diese Malware zeichnet sich z.B. dadurch aus, dass sie als sensibel markierte Daten stiehlt, verfügbar macht oder modifiziert.

## 2.3 Design und Implementierung

Um ein systemweites und feinkörniges Tainting zu ermöglichen, wurde Panorama darauf ausgelegt, eine komplette PC-Hardware samt CPU, Betriebssystem und Anwendungen zu simulieren. Um dies möglich zu machen, setzt es auf QEMU [Bel], einem generischen Open Source Prozessor Emulator, auf.

Um die Markierungen zu speichern, wird ein so genannter Shadow Memory benutzt. Darin werden für jedes einzelne Byte des Systemspeichers alle notwendigen Informationen, um später die Taint Graphs zu erstellen, festgehalten. Dieser Shadow Memory wird auch auf die Harddisks angewandt, um ausgeswappte Speicherseiten und Information, die in Dateien gespeichert wird, zu verfolgen.

Als sensible Information werden alle Informationen, die von den automatisierten Tests in das System gebracht wurden, markiert. Darunter fallen Eingaben von Harddisks, Netzwerk oder Keyboards. Allerdings ist Panorama auch in der Lage, den Output von bestimmten System Calls automatisch als sensibel zu markieren. Generell ist es am besten, die Markierungen so früh wie irgend möglich zu machen, um etwaige Umgehungsversuche von Malware zu unterbinden.

Wenn eine Datenquelle einmal als sensibel markiert ist, dann müssen alle CPU Instruktionen, aber auch DMA Operationen, die diese Daten verwenden bzw. sie manipulieren, auf mögliche Propagierungen der Markierung hin untersucht werden. Dabei werden auch Spezialfälle wie Constant Functions, wie z.B. `xor eax,eax` und table lookups abgedeckt. Letztere werden dabei auf eine konservative Art und Weise behandelt: Wann auch immer ein markierter Wert bei einer Adressberechnung eine Rolle spielt, wird auch das Ergebnis markiert. Einen weiteren Spezialfall stellen so genannte Control Flow Evasions dar. Dafür bietet Panorama derzeit keine automatische Vorgehensweise. Bekannte Stellen in Betriebssystemen müssen manuell überarbeitet werden, um Daten zu markieren. In Zukunft wäre es möglich, für diesen Fall eine statische Analyse in Panorama einzubauen.

Zusätzlich zu hardware-nahem Tainting müssen auch Informationen des Betriebssystems in Betracht gezogen werden, denn es ist wichtig, zu wissen, von welchem Prozess sensible Daten modifiziert bzw. benutzt werden. Zum Zwecke dieser Informationsbeschaffung vom Betriebssystem wurde ein eigenes Kernel Modul entwickelt. Interessant ist weiters auch die Information über Files, mit denen interagiert wird. Dazu wurde das Sleuth Kit [Car07] angepasst und in Panorama integriert.

Der eigentliche Sinn hinter dem Markieren ist die Erkennung von Zugriffen des potentiell bösartigen Programms auf markierte Daten. Dieses Programm könnte nun aber versuchen, Daten nicht selbst zu manipulieren, sondern sie manipulieren zu lassen. Dies kann entweder durch dynamisch generierten Code oder durch den Aufruf von bestehendem Code in anderen Programmen passieren. Ersteres wird durch das spezielle Markieren des Code Segmentes des bösartigen Programms erreicht. Dies ermöglicht es dann, auch alle davon erzeugten Instruktionen als gefährlich zu markieren. Zweiteres wird durch Verfolgen des Stack Pointers erreicht. Konkret werden alle Calls, die von dem bösartigen Code gemacht werden, wieder als gefährlich markiert. Aufrufe von asynchronen Kernel API Funktionen werden als Spezialfall behandelt und dadurch auch abgedeckt.

## 2.4 Automatisches Testen und Taint Graphs Erzeugung

Die Test Scripts für die automatischen Tests unter Windows XP wurden mit AutoHotkey [Mal] geschrieben. Diese freie Software erlaubt es, Scripts als Text Dateien zu erzeugen oder aber Eingaben aufzuzeichnen. Es gibt verschiedene Typen von Taint Source. Derzeit sind in Panorama folgende implementiert: text, password, HTTP, HTTPS, ICMP, FTP, document und directory. Beim Propagieren der Markierungen ist es wichtig, dass Test Scripts die Möglichkeit haben, Metainformationen über die Inputdaten an die Taint Engine weiterzugeben. Dies wird durch spezielle Windows Registry Einträge ermöglicht.

Die Taint Engine erzeugt genau einen Graph für jeden Taint Source mit einem speziellen Typ. Beim Erzeugen dieser Graphen wird die hardware-nahe Information auf eine Betriebssystem-spezifische umgelegt. Auf dieser Ebene wird zwischen Prozessen, (Kernel)Modulen, Netzwerk und Dateien unterschieden.

## 2.5 Malware erkennen und analysieren mittels Taint Graphs

Derzeit unterscheidet Panorama zwischen drei verschiedenen bösartigen Verhaltensmustern. Unter abnormalem Informationszugriff versteht man einfache Leseoperationen des getesteten Programms auf als sensibel markierte Daten. Ein Beispiel für ein bösartiges Programm in dieser Kategorie wäre ein Backdoor, welches Befehle übers Netzwerk entgegennimmt. Das zweite Verhaltensmuster sind anormale Informationsdistributoren. Das betrifft Programme, die bestimmte Daten zwar lesen. allerdings nicht weiterleiten dürfen. Ein Beispiel dafür sind Browser Help Objects (BHOs). Zuletzt gibt es noch Programme, die exzessive Informationsbeschaffung betreiben. Dies sind Programme, deren Ausmaß an Zugriffen auf Daten eine Bösartigkeit ausmacht. Ein Beispiel dafür wären Rootkits, welche typischer Weise überaus viele Zugriffe auf Verzeichnisinformationen tätigen.

Neben der automatischen Erkennung von Malware ist auch die manuelle Analyse für Systemadministratoren und Sicherheitsbeauftragte sehr wichtig. Diese können anhand eines generierten Taint Graphen einfach Rückschlüsse über die Funktionsweise von Malware ziehen.

## 2.6 Evaluation von Panorama

Das fünfte Kapitel des Panorama Artikels befasst sich ausführlich mit den durchgeführten Tests. Über diese Tests kann aus Platzgründen hier nur ein kurzer Überblick gegeben werden. Es wurden 42 Malware Beispiele aus dem Internet, der Forschung und eines österreichischen Antivirus Herstellers für Tests herangezogen. Darunter finden sich 5 Keylogger, 2 Passwort Klauer, 2 Netzwerk Sniffer, 3 Stealth Backdoors, 22 BHOs und 8 Rootkits. Weiters wurden 56 gutartige Programme untersucht. Dabei wurden keine False Negatives und nur sehr wenige False Positives entdeckt, zwei davon waren Personal Firewall Programme.

Die Performance von Panorama ist kein primäres Ziel gewesen, da das Tool nicht für den Hintergrundeinsatz gedacht ist. Trotzdem wurde sie gemessen und der Verzögerungsfaktor in der Ausführung liegt in etwa bei 20.

## 2.7 Diskussion der Methoden

In diesem Kapitel werden denkbare Umgehungsmaßnahmen von Malware diskutiert und mögliche Gegenmaßnahmen angedacht. Es ist denkbar, dass ein Malware Programm versucht, die Taint Engine zu überlisten, indem die Propagierung der Markierungen verhindert wird. Derzeit ist es durch fehlende statische Analyse möglich, dass ein Schadcode sensible Informationen weiterleitet, ohne erkannt zu werden. Allerdings ist es nicht möglich, dass ein Schadcode sensible Daten lesen kann, ohne erkannt zu werden. Das reicht im Normalfall aus, um ein Programm als Malware zu klassifizieren.

Dankbar wäre auch die Umgehung einer Erkennung durch nicht Ausführen von bösem Verhalten bis gewisse Bedingungen erfüllt sind wie z.B. ein Timeout abgelaufen ist, oder bis keine QEMU Emulation mehr erkannt wird. Gegen zweitere Umgehungsversuche helfen Stealth-Technologien, wie in [VY06] beschrieben. Solche Mechanismen wollen die Autoren in Zukunft in Panorama integrieren.

Ein weitere Möglichkeit von Malware wäre es, Panorama selbst mittels z.B. eines Buffer Overflows anzugreifen und zu manipulieren. Dies kann nur durch einen Bug in Panorama passieren. Die Lösung dieses Problem wäre in diesem Fall das Beheben des Bugs.

## 2.8 Zusammenfassung

Malware bringt ernste Gefahren für Sicherheit und Privatsphäre mit sich. Panorama ist ein systemweites, feingekörntes und auf Tainting basierendes Framework, welches Verhaltensweisen von unbekanntem Code erkennen kann. Die Funktion von Panorama wurde durch ausführliche Tests belegt. Die Autoren von Panorama glauben, dass es eine unabdingbare Hilfe für die Analyse von potentieller Malware darstellt.

# 3 Vergleich mit anderen Arbeiten

Im Vergleich zu dem in der Einführung beschriebenen Cross Site Scripting Prevention Tool ist Panorama nicht für den ständigen Betrieb im Hintergrund ausgelegt. Dies wäre auch nicht effizient möglich, da es Programme in etwa um den Faktor 20 verlangsamt. Panorama ist als Hilfe zur Analyse von unbekanntem Code gedacht, der, sobald für gut befunden, dann ohne bzw. mit herkömmlichen Schutzmechanismen ausgeführt werden kann.

Trotz dieses fundamentalen Unterschiedes gibt es etliche Gemeinsamkeiten. Beide Ansätze sind auf Initiative des Endbenutzers einer Applikation einsetzbar, er ist nicht länger auf die Sicherheitsbemühungen von z.B. Serverbetreibern angewiesen. Beide Ansätze verwenden dynamisches Tainting, um sensible Daten zu markieren und diese Markierungen zu propagieren. Statisches Tainting ist auch im Konzept von Panorama enthalten und muss nur noch implementiert werden.

Arbeiten, die nicht auf Tainting aufgebaut sind, benutzen zumeist entweder Heuristiken oder Signaturen. Signaturen haben den großen Nachteil, dass keine neuen Bedrohungen erkannt werden, sondern ausschließlich bereits zuvor bekannte. Heuristiken haben dieses Problem zwar nicht, aber wie in Kapitel 2.1 beschrieben, liefern sie naturbedingt viele Falschmeldungen.

# Literatur


[Bel] F. Bellard, "Qemu, a fast and portable dynamic translator," pp. 41–46. [Online]. Available: http://www.usenix.org/publications/library/proceedings/usenix05/tech/freenix/bellard.html

[Car07] B. Carrier, "The sleuth kit." http://www.sleuthkit.org/sleuthkit/desc.php, July 2007. [Online]. Available: http://www.sleuthkit.org/sleuthkit/desc.php

[Mal] C. Mallet. AutoHotkey. [Online]. Available: http://www.autohotkey.com

[VNJ+07] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," in *Proceeding of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2007.

[VY06] A. Vasudevan and R. Yerraballi, "Cobra: Fine-grained malware analysis using stealth localized-executions," *sp*, vol. 0, pp. 264–279, 2006.

[YSE+07] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security.* New York, NY, USA: ACM, 2007, pp. 116–127.