

OMCD

Open Methodology for Compromise Detection v. 0.2. (Windows Systems)

omcd<at>isecom.org

June 2006

Project Managers

Christopher Schooley
Pete Herzog

Goals

The OMCD project defines a methodology to accurately detect if a given system has been compromised, the extent of compromise, and the specific techniques used. The methodology is intended as a guide for security analysts and researchers to mitigate or analyze system compromise techniques, and as a aid and to system developers designing new systems or security controls.

The OMCD addresses both persistent changes to a compromised system and volatile or state-based conditions that, without proper care, could be lost before analysis is complete. The document discusses both conventional compromises and rootkits. The OMCD project documents are specific to the Operating System for which they were written. Ensure you download the appropriate version (the OS will be in the title).

Assumptions

No baseline information assumed

This will ensure that the methodology can be applied to all systems, not only those which were specially prepared for future investigation. Because of this assumption, some popular, anti-compromise solutions, like for example hash based file system integrity checking, will not be discussed.

No signature based detection techniques

All signature based detection techniques has common drawback of not being able to deal with unknown attacks, even if the attacks do not use any original techniques. Thus, the methodology described in this document does not discuss signature based approaches. This does not preclude the use of hash checks against "known good" system files.

Applicability

The aim of the OMCD is to develop methodologies that apply to all common operating systems and system types. Due to technical implementation details OMCD documents are tailored to a specific OS and grouped in accordingly into OMCD Tracks.

This particular OMCD Track (Windows 2000/2003/XP) focuses on Windows 2000, XP and 2003 family of Windows systems. While much of the theory and approach are common to all Tracks many of the techniques are OS specific.

1 Introduction

- 1.1 Definition of System Compromise
- 1.2 Goals and Assumptions
- 1.3 About this Document
- 1.4 Preserving System Information
- 1.5 The Iterative Process of Compromise Detection (flowchart)

2 Applying The OSTMM Channel Map

- 2.1 Physical
- 2.2 Data Networks
- 2.3 Wireless Communications
- 2.4 Telecommunications
- 2.5 Personnel

3 Network Activity

- 3.1 Network and Telecommunications Interfaces
- 3.2 Capture/Analyze Traffic
- 3.3 Listening Ports
- 3.4 Active Sessions
- 3.5 Routing Table & ARP Cache
- 3.6 Software Firewall & IPSec Policies
- 3.7 Categorizing Traffic Patterns
 - 3.7.1 Classic Back door Traffic
 - 3.7.2 Covert Channels Traffic
 - 3.7.3 Passive Covert Channels in Legitimate Traffic

4 File System Verification

- 4.1 Verifying Core System Files
 - 4.1.1 Verifying Certificate Store
 - 4.1.2 Using Heuristics (AV Like Approach)
- 4.2 Discovering suspect files
- 4.3 Detecting hidden files & NTFS streams

5 Registry Verification

- 5.1 Analyzing Registry Permissions
- 5.2 Discovering Suspect ASEPS

- 5.3 Discovering Other Suspect Keys
- 5.4 Discovering Hidden Keys
 - 5.5 Hidden via Unicode strings
 - 5.6 Hidden via long strings
- 6 Running Processes**
 - 6.1 Enumerating running processes
 - 6.1.1 Finding hidden processes
 - 6.2 Identifying running processes
 - 6.3 Analyzing process list
- 7 Services**
 - 7.1 Analyzing Service Settings
 - 7.2 Running vs. Stopped/Unresponsive Services
 - 7.3 Parameters passed at service startup
 - 7.4 Security Context of Services
- 8 Permissions and System Security Context**
 - 8.1 Identifying Suspect User Accounts
 - 8.2 Mapping Group Memberships
 - 8.3 Audit Permissions Applied to Objects
 - 8.4 Audit User/System Policies
 - 8.4.1 Application of GPO
 - 8.5 Audit File and Print Shares
 - 8.6 Verify Domain Membership
- 9 Application integrity verification**
 - 9.1 General Methods
 - 9.2 Analyzing Application's Relation to OS
 - 9.2.1 Case Study: Internet Explorer & Components
 - 9.2.2 Case Study: Firefox & Components
- 10 Rootkit Detection & Memory Verification**
 - 10.1 Usermode-level memory verification
 - 10.1.1 Discovering suspected (not hidden) processes and threads
 - 10.1.2 Code sections verification
 - 10.1.2.1 Verification against image files
 - 10.1.2.2 Heuristic based hooks detection
 - 10.1.3 IAT/EAT tables verification
 - 10.1.4 Considerations for memory access techniques

