REcon 2016: Reverse Engineering Everything

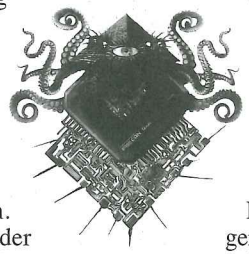
Deep Inside

Simon Metzler

Wenn sich die Mehrheit der langhaarigen Besucher einer Veranstaltung in die Warteschlange zur Herrentoilette einreiht, ist man entweder auf einem Metal-Konzert oder auf einer Hacker-Konferenz. Die REcon in Montreal gehört zu Letzterem.

Unter den vielen Konferenzen im IT-Sicherheitsbereich zählt die REcon mit rund 400 Teilnehmern zu den kleineren. Besetzt ist sie allerdings hochkarätig, und viele Persönlichkeiten aus der Reverse-Engineering- und IT-Forensik-Szene nutzen die Gelegenheit zum fachlichen Austausch und Networking. Die Bandbreite der Themen reicht von neuen Tools/Frameworks, die das Leben von (Malware) Reverse Engineers erleichtern sollen, über zahlreiche Geschichten, wie bestimmte Produkte analysiert und gebrochen wurden, und die Ergebnisse von im Zuge forensischer Untersuchungen analysierter Malware bis hin zu fundierten Aufarbeitungen bestimmter Mechanismen in aktuellen Windows-Systemen.

Im Vortrag von Alex Ionescu, Mitautor des Buches „Windows Internals", ging es um die Kernel Shim Engine (KSE), einem nicht dokumentierten Feature in aktuellen Windows-Systemen, die das Hooking von Windows-Treibern erlaubt. Aus sicherheitstechnischer Sicht sind Hooking und Shimming allgemein Techniken, die besonders für Malware- beziehungsweise Rootkit-Autoren interessant sind, um sich tief im System einzunisten. Damit das nicht jeder kann, gibt es eine Datenbank registrierter Shims – allerdings werden nicht alle genutzt und deren „Identität" lässt sich daher „spoofen". Alternativ reicht es, schneller als ein legitimer Shim zu sein, der bewusst verzögert geladen wird. Auch der Austausch der Shim-Datenbank (SDB) durch eine manipulierte Version ist denkbar, wenn diese nicht signiert ist. Im Umkehrschluss bedeutet dies, dass sich forensische Analysen an Live-Systemen künftig auch auf diesen Aspekt konzentrieren müssen. Ein Tool dafür – DriverMon – wurde bereits angekündigt.



Gefährliche Optimierungen

Eine wichtige Lektion gab es von Robert C. Seacord, Professor der Carnegie Mellon University in Pittsburgh (USA) und beteiligt an der Standardisierung von C. Um Letzteres ging es auch in seinem Vortrag. Compiler – die Beispiele im Vortrag bezogen sich auf die GNU Compiler Collection (GCC) – sollen eine Programmiersprache möglichst effizient in Maschineninstruktionen übersetzen. Das beginnt mit der Einsparung unnötiger Variablen und nicht erreichbaren Codes, geht über die speichereffiziente Anordnung von Instruktionen und Daten sowie den Ersatz ineffizienter Instruktionen durch bessere und endet bei der Ausnutzung nicht definierten Verhaltens.

Dass der heutige Trend „Total License", bei dem nicht definiertes Verhalten als einzusparender toter Code gewertet wird, gestoppt werden sollte, hat Seacord anhand eines Beispiels erläutert. Ein durch den Programmierer vorgenommener Fix eines regulären Bugs resultierte in einem, aus Sicht des Compilers, immer noch nicht definierten Zustand, woraufhin der Compiler eine sicherheitskritische *if*-Anweisung kürzte. Er empfahl daher, dass Entwickler sicherheitsrelevante Flags des Compilers nutzen – im Fall des GCC sind das *-O2* oder *-O3* sowie *-Wstrict-overflow* – und den Code gegebenenfalls anpassen.

Dass das sicherheitstechnisch korrekte Übersetzen von Code schwierig ist, hat auch Natalie Silvanovich anhand von Flash dargelegt. Hier wird der ActionScript-Code gleich zweimal übersetzt: Zuerst in Byte-Code und dann erst in konkrete Instruktionen. Ihre Erkenntnis nach vielen gefundenen Fehlern lautet, dass sich Bugs nicht nur durch Reversing mit IDA, sondern auch durch kreatives Fuzzing aufspüren lassen – wenn auch mit geringerer Exploitability. Die gute Nachricht dabei ist, dass die Exploits allesamt kein so kompiliertes ActionScript sind, sondern explizit manipulierter Byte-Code. (ka)