Obsolescenza tecnologica; conoscerla per combatterla con efficacia

Author : Massimiliano Brolli

Date : 10 ottobre 2018

Molto spesso mi viene chiesto come poter sostituire il software obsoleto presente sulle applicazioni evitando gli elevati impatti economici, le riscritture di codice e soluzioni custom. Purtroppo non c'è una risposta o una soluzione definitiva, in quanto tutte le cose (e anche le applicazioni) hanno un loro ciclo di vita. È però possibile dare delle indicazioni su come trattare in modo efficace il fenomeno dell'obsolescenza, cosa che tenterò di fare con questo articolo.

Cosa è l'obsolescenza tecnologica?

È un fenomeno derivato dal fatto che una componente software (ad es. un sistema operativo, un framework di sviluppo, ma anche il firmware di uno smartphone o di una stampante) ha un periodo di supporto da parte del fornitore limitato nel tempo, che di solito risulta sufficientemente lungo. Superato questo periodo il fornitore cessa la fornitura di aggiornamenti e bug-fix (comprese le patch di sicurezza) causando l'accumulo e la stratificazione delle vulnerabilità nel tempo e rendendo i nostri sistemi facilmente violabili.

Si tratta di un rischio che consente vettori di attacco molto pervasivi come ad esempio i pericolosi RCE (Remote Code Execution) capaci di compromissioni e impatti di ogni natura e profondità. Purtroppo, spesso, la via più immediata per trattare un sistema obsoleto è riscriverlo o riadattarlo alle più recenti componenti software. Su applicazioni enterprise questo può risultare altamente costoso oltre a non dare vantaggi agli utenti in termini di CX. Ciò costituisce un ostacolo importante, che spesso porta l'utente a dire *"sono pienamente soddisfatto del mio sistema e non vedo il motivo di dover spendere dei soldi per riaverlo esattamente come prima".*

Siamo realisti, non è facile trasmettere questi concetti ad utenti non esperti di sicurezza. Per mia esperienza aiuta molto dimostrare ciò che è possibile fare sfruttando l'obsolescenza tecnologica; l'utilizzo di messaggi diretti come: *"Sottrazione di tutte le carte di credito degli utenti da rete internet in accesso anonimo a causa dell'obsolescenza tecnologica*", vi assicuro che vale più di altre mille altre parole.

Addentriamoci nel merito, come sempre la sicurezza passa per i soliti due concetti ovvero la "*Security by Design*" e "*Long-Term Security*" e questo fenomeno, ovviamente non è da meno.

Security by Design

- **Scegliere il software con attenzione:** nei nuovi progetti, porre massima attenzione a tutta la pila software utilizzata, in modo che sia opportunamente aggiornata e garantisca tempistiche di supporto adeguate. Ad esempio, se oggi avviamo un nuovo progetto su tecnologia Microsoft potremmo utilizzare SQL Server 2008 R2 (tuttora a listino) che va in end-of-life a settembre 2019. Una svista di questa natura è "clamorosa" in quanto tra meno di un anno tale release andrà in end-of-life e non verranno più rilasciate patch di sicurezza. Scegliendo invece SQL Server 2017, il problema verrà spostato a dicembre 2027, ben 9 anni di supporto in più, un’eternità nella gestione di un sistema.
- **Non far decidere agli altri, decidete voi:** molto spesso i fornitori tendono a creare sistemi su tecnologie datate per evitare costi di formazione su quelle più recenti. Occorre prestare la massima attenzione a questo fenomeno (in particolare su software open-source) in quanto non è raro avere nuovi sistemi con software già in end-of-life non appena questi varcano la soglia dell'esercizio.
- **Componenti Open Source:** fate molta attenzione. A parte il software Open riconosciuto a livello internazionale (ad es. MySql, Apache, Tomcat, Java, ecc..) con vendor o community di spessore alle spalle, scegliete le librerie e i framework che notoriamente non hanno problemi di sicurezza ricorrenti (ad es. Drupal, Joomla, Struts2, ecc...). Verificate anche che le relative community risultino attive prima di adottarle nella vostra architettura software, in quanto utilizzare software open-source con community in decommissioning porterà ad una veloce obsolescenza, e questo lo dobbiamo evitare.
- **Framework e librerie open source & buy:** come visto in precedenza se l'applicazione è scritta da noi, utilizzare il più possibile codice proprietario e librerie messe a disposizione dai framework di sviluppo (ad es. librerie native di Java, Namespaces Microsoft .NET) evitando di utilizzare componenti di terze parti per effettuare operazioni implementabili in modo “semplice”. Ad esempio, se in ambiente .NET ci occorre una libreria che invia mail, non acquistiamo una DLL di terze parti ma sviluppiamo le nostre classi estendendo il namespace "System.Net.Mail.SmtpClient". Insomma, più autonomia e controllo abbiamo sul codice da noi prodotto, più facile risulterà agire.
- **Patching automatizzato:** per tutte le tecnologie che lo consentono (ad es. Prodotti Microsoft o Red Hat Satellite) abilitare il patching automatizzato e valutare, in caso di tecnologie che non lo consentono, prodotti di terze parti da implementare.

Long-term Security

- **Vulnerability Assessment ricorsivi:** Effettuare scansioni cadenzate sulle applicazioni più critiche monitorando lo scostamento nel tempo (recycle) delle vulnerabilità sfruttabili ed effettuare tutte le mitigazioni del caso. Alle volte sono presenti servizi non necessari che introducono vulnerabilità gravi che se correttamente segregati possono mitigare o risolvere completamente il problema.
- **Patching delle componenti obsolete:** per le componenti obsolete, svolgere il patching

fino all'ultima versione rilasciata dal fornitore. Questo garantisce una precisa identificazione delle potenziali minacce in fase di Assessment che, abbinata alle scansioni ricorsive, consente di tracciare un quadro ben preciso delle reali vulnerabilità presenti sui sistemi.
- **Patching automatico:** a regime, come visto nella security by design
- **Rimozione o segregazione dei servizi non necessari:** attivare i soli servizi necessari al funzionamento del sistema. Per quelli che risultano necessari ma vulnerabili, ridurre il più possibile l'esposizione all'esterno. Ad esempio, se abbiamo OS Windows 2000, il servizio RDP (altamente vulnerabile a RCE) deve essere reso disponibile ai soli IP/subnet/VLAN/interfacce necessarie all'amministrazione del servizio. Così facendo ridurremo drasticamente la superfice di rischio oltre ad identificare con precisione i possibili potenziali aggressori.
- **Valutare sistemi di sanificazione degli input:** qualora non sia possibile (o altamente costoso procedere al replatforming) valutare la possibilità di messa in rete su applicazioni critiche o esposte su internet di sistemi WAF di 3rd generazione / Virtual Patching. Questi sistemi consentono di limitare le richieste anomale in generale e possono essere molto utili per arginare l'obsolescenza tecnologica evitando l'iniezione di payload dannosi. Ricordiamoci che ogni applicazione è unica e occorre analizzare con attenzione i falsi positivi e negativi introdotti da questo genere di tecnologie sul sistema target.

Terminata la "Security by design" si entra nella "Long-term security" e alla successiva fase di gestione dell'obsolescenza.

Conclusioni

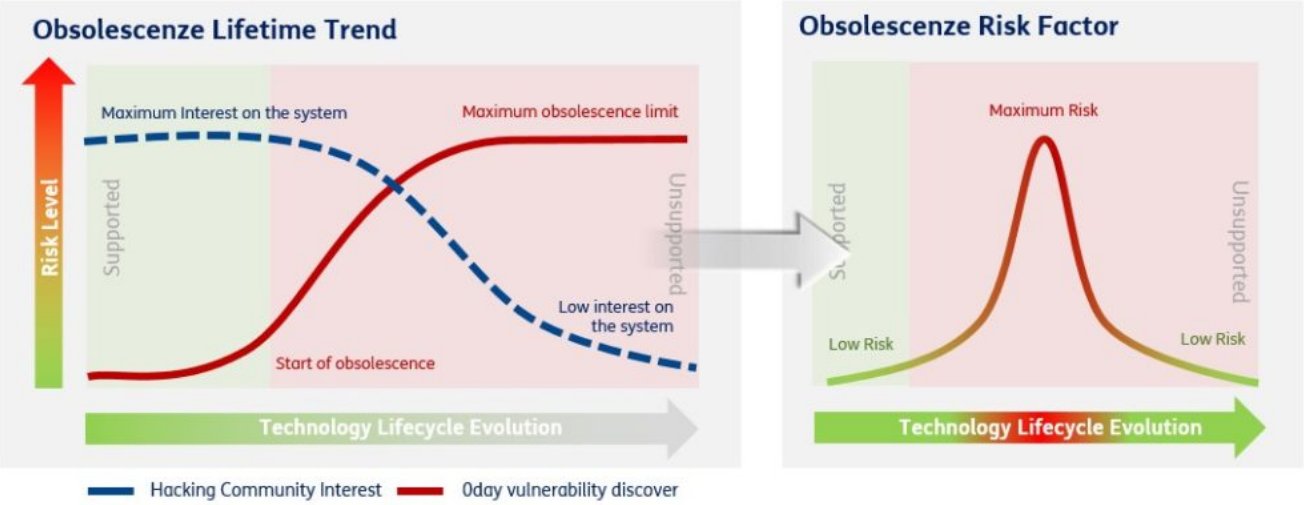
L'obsolescenza tecnologica è come la data di scadenza di una medicina. È una minaccia che, anche se oggi non è presente, prima o poi si manifesterà. Errori grossolani di progettazione sulle nuove applicazioni portano ad avvicinare frettolosamente questa data, quindi occorre prestare massima attenzione ai nuovi sistemi.

La strada del replatforming (qualora possibile) risulta sempre la migliore, ma sulle applicazioni legacy/enterprise già obsolete non sempre è percorribile, a causa dei massicci investimenti richiesti. Qualora non sia possibile un buon "cleaning" (hardenizzazione e messa in sicurezza delle vulnerabilità rilevate) può consentire una drastica riduzione - e alle volte una completa bonifica - del rischio dovuto all'obsolescenza tecnologica. Questo intervento (normalmente di basso costo) dovrà essere eseguito "*da mani esperte*" in quanto può bastare una sola vulnerabilità per rendere vano l'intero lavoro svolto.

E' inoltre buona norma una volta completata l'attività di "cleaning", verificare con cadenza ricorsiva (almeno una volta all'anno) la presenza di ulteriori vulnerabilità sfruttabili, per evitare l'"accumulo" e la "stratificazione" che l'obsolescenza genera nel tempo.

In tutto questo, ricordiamoci sempre che le applicazioni sono "allergiche" agli aggiornamenti software e che seppur sia corretto aggiornare i sistemi con cadenza regolare (rischio di compliance per tipologie di dati trattati) non è sempre possibile farlo per ragioni di servizio e di

costo; conviene spesso concentrare le forze nella risoluzione puntuale delle minacce rilevate sugli host vulnerabili. Non sottovalutiamo anche il fatto che alcune volte le patch di sicurezza sui sistemi supportati risolvono vulnerabilità pregresse ma ne inseriscono di nuove (ad esempio la CVE-2017-10271 su Oracle, che introduceva una pericolosa RCE e DOS non presenti nella precedente CPU).
Inoltre - e paradossalmente - passato un certo tempo dall'inizio dell'obsolescenza si assiste ad un abbattimento del livello di rischio per una carenza di interesse nelle community underground e per assenza di letteratura sulle vulnerabilità presenti così come viene mostrato nel grafico di seguito.



Concludendo, se la "security by design" non è stata svolta o siamo entrati in obsolescenza è possibile agire efficacemente sul sistema ma questo deve essere fatto da "*mani esperte*"; resta a noi valutare con precisione gli skill di chi supervisionerà questo genere di attività decisamente molto più vicine all'offensive-security che al mondo sella Security high-level.

Rendere sicuro un sistema obsoleto è possibile ma richiede una grande precisione e attenzione sui dettagli. Purtroppo tutto questo si scontra con una grande carenza negli skill e una assenza di consapevolezza del rischio presente nella piccola, media e grande impresa e su cosa potrebbe accadere quando quella minaccia si trasformi in compromissione, ma questo è un altro problema.

Articolo a cura di **Massimiliano Brolli**