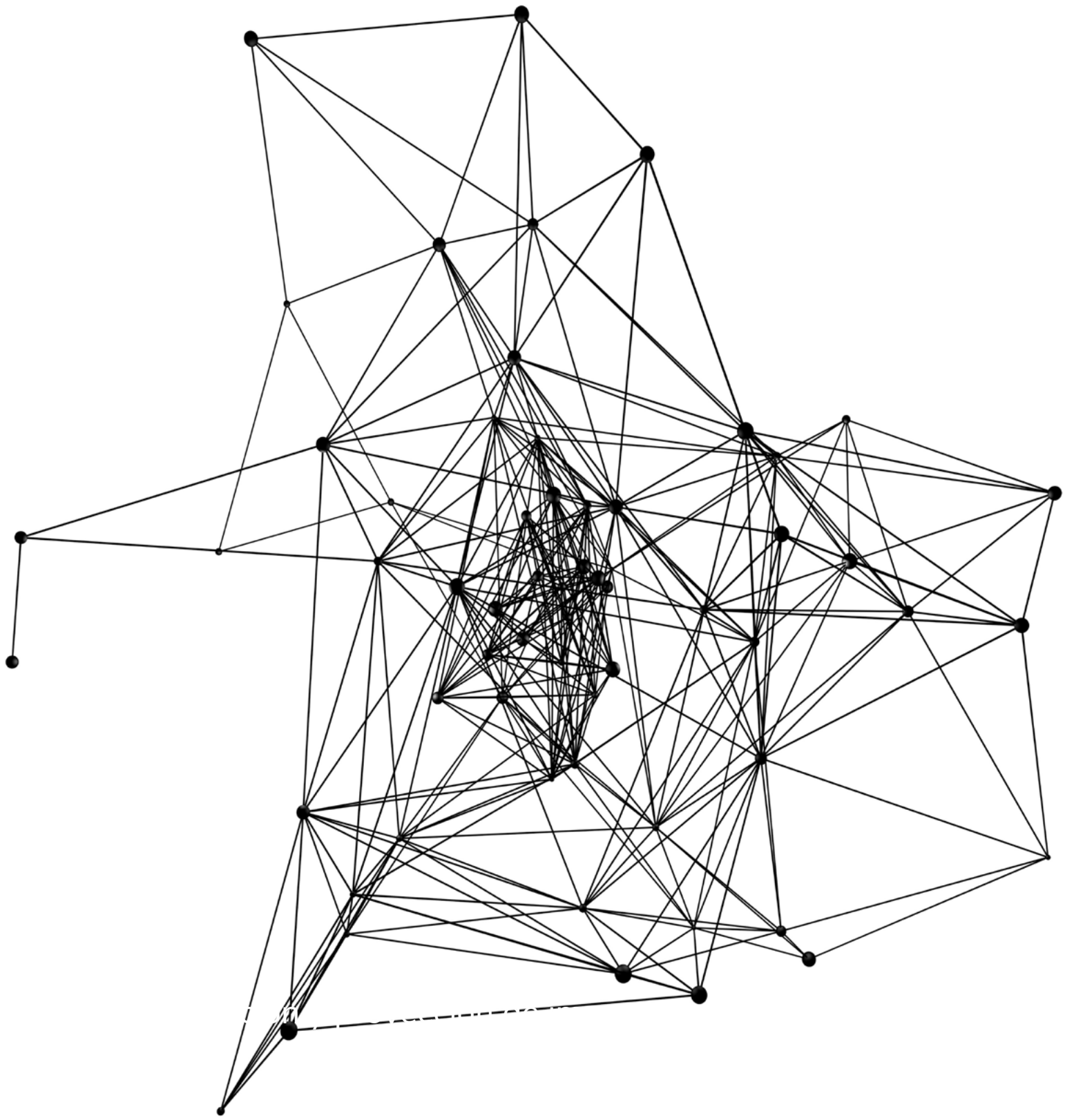
THREATSCOUT®

N.05
Abril, 2021

La interrelación entre complejidad y seguridad (parte I)



“Everything should be made as simple as possible, but not simpler.” - Albert Einstein

Joseph Tainter sostiene que, a lo largo de la historia, las sociedades han colapsado cuando se vuelven demasiado complejas e insostenibles en la medida en que evolucionan y buscan soluciones a sus problemas. También menciona que el ritmo de innovación actual no es sustentable. En ningún momento menciona directamente la situación del software hoy en día, mucho menos en lo que respecta a su calidad (o falta de).
En esta publicación vamos a hacer un ensayo que busca unir algunos de estos argumentos con la situación actual referente a la ciberseguridad, con énfasis en la del software (lo que en inglés denominan “AppSec”).

Si usted hace 30 ó 40 años atrás programó en lenguaje ensamblador para un microprocesador como el 6502, estaba en una situación bastante más favorable que cualquier otra persona que hiciera lo mismo programando en lenguaje de máquina o utilizando opcodes.

A su vez, otras personas que tuvieran acceso a programar en lenguajes como BASIC, disfrutaban de una experiencia más productiva (en términos de tiempo) y menos dolorosa (en términos de complejidad) pero quizá menos potente (en términos de performance) ó poderosa (en términos de control directo del hardware) que hacerlo en lenguaje ensamblador. La creación de un lenguaje como BASIC implica agregar capas de abstracción entre el programador (humano) y la máquina, con la finalidad de reducir la complejidad de interactuar directamente con la tecnología subyacente. La palabra clave es abstracción de la complejidad; todo para que el programador se beneficie de una aparente simplificación. Esto a un costo, como todo en la vida: Una sintaxis de programación simple implica mayor popularidad de la tecnología entre los programadores y los usuarios de sus creaciones (programas ó aplicaciones). A mejores programas creados para esa tecnología, más se impulsan sus ventas (it's the software, stupid). Mientras que el costo para un programador (en forma de menos control sobre el hardware y un poco menos de rendimiento al ejecutar su programa) - represente una mayor productividad y un mejor time-to-market de sus productos, todo está bien.

Regresando a tiempos modernos: hoy en día sería casi impensable plantearnos programar un navegador web en lenguaje ensamblador, mucho menos un conjunto de aplicaciones como Word o Excel. También resulta prácticamente ridículo pensar en crear un portal web utilizando Perl y CGI-BIN, o peor, desarrollar un front-end con JavaScript puro sin emplear algo como JQuery o Bootstrap. Estamos empezando a ver un patrón: necesidades del mercado de crear software más rápido, simplificación de interacción con la tecnología para reducir la barrera de entrada para nuevos programadores, creación de más capas de abstracción para lograr estos fines. En fin: Evolución. ¿Están de acuerdo?. Dicho esto, el costo asociado a la "evolución" está en incremento continuo: cada capa que conforma la abstracción requerida para construir software "moderno" en menor tiempo, es más compleja y se encuentran cada vez más entrelazadas entre sí. A pesar de que existe más información disponible públicamente en forma de estándares, especificaciones técnicas, conceptos y hasta de código fuente de estos programas, cada vez son menos las personas que realmente entienden como funcionan las cosas.

Evidentemente esto es algo natural dado a la complejidad (y a lo que muchos denominan sobre-ingeniería) de la tecnología y en especial del software. La necesidad de especialización es cada vez más latente cuando se requiere profundizar en algún componente específico o en diagnosticar (hacer "troubleshooting") cuando algo contenido dentro de esas capas de abstracción no está funcionando correctamente. Y es ahí precisamente, donde basamos nuestro ensayo de interconectar los problemas actuales de seguridad y la complejidad del software...

La complejidad de construir un programa hoy en día no está tan asociada al conocimiento de la sintaxis del lenguaje de programación que seleccionemos (para eso existe StackOverflow también, ¿no?), sino a como utilizar los "Frameworks", librerías, APIs o web services que vamos a requerir para cumplir con nuestra meta en el menor tiempo posible. Se supone que esos componentes ya fueron probados por alguien más. O quizás no...

La respuesta puede ser agridulce, y si juzgamos por la cantidad de "bugs" y problemas de seguridad que se reportan diariamente en los distintos CVEs (Common Vulnerabilities and Exposures), la respuesta es más amarga y preocupante que dulce. Y he de recordarles que cuando aparece un CVE ya es públicamente conocido el problema... Todavía hay muchas otras vulnerabilidades o defectos que son de conocimiento privado de los fabricantes y los mantienen en silencio. Peor aún: muchos de estos son descubiertos por algún grupo criminal o por alguna agencia de inteligencia, y los reservan para su eventual explotación (vulnerabilidades conocidas como 0-day ó de día cero).

Desde hace casi dos décadas, los principales fabricantes de software empezaron a tomar seriamente en cuenta sus deficiencias en lo que respecta la calidad de sus productos. Las implicaciones negativas de sufrir ataques sobre estos o que mediante su abuso los productos faciliten la ejecución de ataques en contra de clientes del

fabricante o de algún tercero, se sintieron rápidamente en la mala percepción por parte no solo de especialistas del área sino por el público en general que de forma directa o indirecta se convirtieron en víctimas de estos problemas de seguridad.

Las iniciativas desde entonces han sido varias: mejoras en el ciclo de vida de la producción del software, metodologías complejas de "ingeniería" de software, estándares de validación y calidad, entre muchas otras. Han nacido productos para ayudar a la programación segura de aplicaciones, y se han redoblado los esfuerzos para mejorar la calidad y seguridad no solo por parte de los grandes fabricantes, sino en muchas de las empresas que producen software tanto para consumo interno como comercial y/o masivo, sino también por parte de las que prestan servicios de desarrollo a terceros.

Sin embargo, esto no ha desacelerado el flujo continuo de vulnerabilidades ni ha reducido los ataques a productos o servicios basados en software. El agravante en estos tiempos, contrario a las décadas de los 80 y 90, es que el software es ubicuo: dispositivos móviles, sectores salud y finanzas, entretenimiento, aviación comercial y por supuesto: comercio electrónico. Una vulnerabilidad puede resultar en un ataque tipo "Ransomware" que afecte a un hospital y terminen personas falleciendo por esta acción. Un problema de software puede potenciar una falla que termine en una tragedia aérea, como en el caso del 737 MAX de Boeing.

Ahora extrapolemos esta situación unos 10 o 20 años hacia el futuro (un plazo relativamente mediano) e imaginemos las implicaciones de situaciones de esta índole en un mundo prácticamente dominado por algoritmos (no vamos a especular acerca de AGI - Artificial General Intelligence) que de por sí hoy por hoy ya tienen el poder de afectar nuestras vidas, y para ese entonces lo tendrán más aún. Las capas de abstracción necesarias para construir el software de esa época futura, serán mayores y más complejas. Muchas tareas estarán automatizadas o delegadas a entidades autónomas basadas en Machine Learning que, dado a su complejidad, aún menos personas tendrán acceso o los conocimientos requeridos para entender sus fundaciones tecnológicas.

En pocas palabras: más complejidad.
"Complexity is the degree to which the design or implementation of a system or its component is difficult to verify and understand"
Dictionary of computer science, engineering and technology. CRC Press; 2017

La complejidad es el grado en el que un diseño ó implementación de un sistema y sus componentes, es difícil de verificar y entender.

Según un trabajo publicado en el International Journal of Software Engineering & Applications (Vol.11, No.1, January 2020) por Mamdouh Alenezi y Mohammad Zarour del College of Computer and Information Sciences del Prince Sultan University en Riad, Arabia Saudita, desglosan la complejidad en tres áreas:
1 - Complejidad de la arquitectura: definida por la alta interconexión entre sub-sistemas.
2- Complejidad "ciclomática": basada en la cantidad de condiciones tipo if y bucles o loop en el software.
3- Complejidad cognitiva: resultante de la observación directa sobre el programador durante su proceso de resolución de problemas, donde se evidencie su falta de conocimiento que lo lleva a introducir defectos en el código; la falta de claridad en alguna de las funciones del programa que resulte en deliberación extensiva por parte del programador y por último en los problemas resultantes por falta de claridad en el código fuente y situaciones subjetivas que pueden exacerbar estos problemas.

También mencionan la complejidad en las entradas - input - que utilizan los programas para operar, como uno de los principales riesgos de seguridad y potenciales culpables en la introducción de vulnerabilidades en un sistema.
En pocas palabras: la complejidad puede ser vista también como la falta de calidad en un proyecto de software. Gary McGraw en su libro Software Security: Building Security In menciona que la complejidad puede ser considerada el principal enemigo de la seguridad.

Sin embargo, otros estudios[1] han fallado a la hora de demostrar una relación directa entre la complejidad y la seguridad. Por otra parte, investigaciones más empíricas llevadas a cabo por parte de miembros de distintos proyectos Open Source , entre ellos la fundación Mozilla responsable del navegador Firefox, sí han logrado identificar algunos elementos importantes que ponen en evidencia la existencia de este vínculo entre la complejidad y la seguridad. Un ejemplo es el trabajo titulado Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities llevado a cabo por Istehad Chowdhury y Mohammed Zulkernine en el cual[2] utilizando una serie de técnicas estadísticas y datos históricos logran predecir vulnerabilidades con bajos niveles de falsos positivos. Esto, basado en el estudio de métricas relacionadas a la calidad del software y su complejidad.

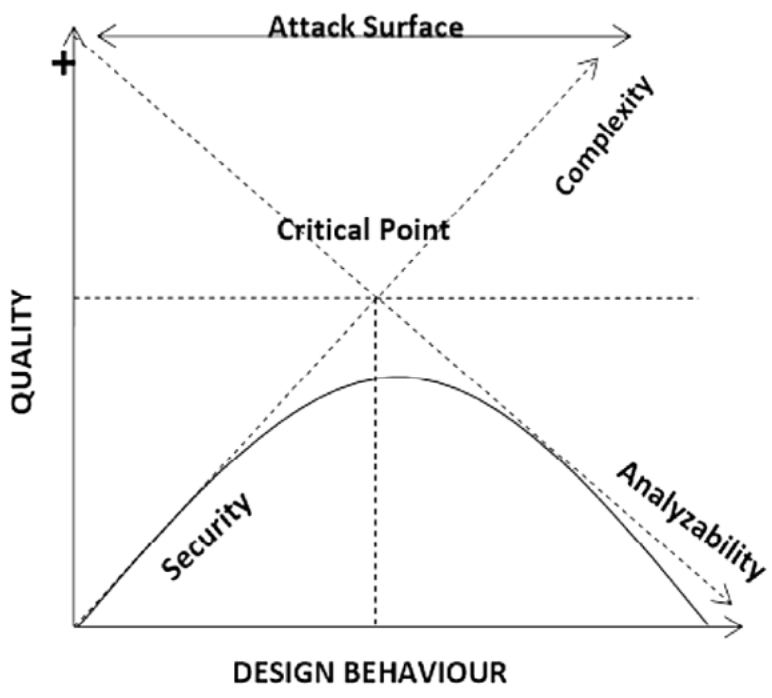
Según las leyes de la evolución del software de Meir Lehman, a medida que un programa del tipo "E" evoluciona, su complejidad también lo hará a no ser de qué se lleven a

1 Challenges with applying vulnerability prediction models https://dl.acm.org/doi/10.1145/2746194.2746198

2 Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities https://bit.ly/2QM7CKO

cabo acciones que lo mantengan o la reduzcan[3]. Esto pareciera implicar que, el hecho de realizar trabajos de mantenimiento sobre el código, reducir su complejidad mediante de-acoplamiento o simplificación de su arquitectura, la situación de seguridad debería mejorar, ¿cierto?.

Figure 1 - Security vs Complexity - Alenezi & Zarour



La realidad es que no es una situación determinística ni que tenga respuestas concretas, fáciles ni claras. Quedan al aire temas como la antes mencionada falta de conocimiento por parte del programador: la complejidad en sí reduce la posibilidad de que alguien pueda tener pleno dominio de las funciones, algoritmos, tecnologías y demás elementos que caractericen a un programa o peor aún: a un sistema.
Por otra parte, la confiabilidad de las herramientas de análisis de código estático es variable y muchas veces útil solo para identificar lo obvio. Esto hace que la necesidad de contar con un par de ojos calificados, minuciosos y críticos sean uno de los principales factores en contra de vulnerabilidades que no sean tan obvias que puedan ser identificadas a tiempo por las herramientas automatizadas.

Por otra parte, también está el factor de que la introducción de vulnerabilidades no esté necesariamente vinculada a negligencia ni a falta de mantenimiento del código: la complejidad y dificultad en la mantenibilidad de este, facilita que alguien pueda introducir vulnerabilidades con propósitos maliciosos. En tiempos recientes, esto se ha puesto de manifiesto en casos como el de SolarWinds y en días atrás en el proyecto del Kernel de Linux, como parte de un experimento mal ejecutado, llevado a cabo por parte de algunos miembros de la Universidad de Minnesota.

En la parte 2 de este trabajo, mencionaremos algunos ejemplos con los que nos hemos topado en revisiones de código pasadas, en los que se demuestra como la complejidad atenta en contra de la seguridad, y de cómo vulnerabilidades introducidas al código, sean estas consecuencia de negligencia o a propósito, pueden ser detectadas únicamente durante una revisión exhaustiva de código.
Por último, dejamos una breve reflexión:
¿Es sustentable mantener el ritmo de innovación actual en la producción de software, tomando en cuenta los crecientes problemas de seguridad y falta de calidad, exacerbados por la incesante exigencia del time-to-market en tiempos cada vez menores?

Como profesionales, pero sobre todo como apasionados de la seguridad, ¡estamos muy interesados en tener estas discusiones contigo!

3 Segunda ley de Lehman: “Increasing Complexity”
https://bit.ly/3xxqfTs

Acerca de Nosotros

Somos un equipo de expertos en ciberseguridad, apasionados por innovar y desarrollar soluciones para contrarrestar retos y riesgos del entorno digital. Creemos en una evolución digital constante que, paralelamente, requiere de capacidades específicas para cerrar paso a quienes hacen el mal.

Nuestra manera de hacer esto, y lo que nos motiva a diario, es poder brindar a empresas la tranquilidad de dejar su ciberseguridad en nuestras manos.

TEKIUM

Avenida Patriotismo #767, piso 7. San Juan Mixcoac, C.P 03730. Ciudad de México.