UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN ESCÁNER DE VULNERABILIDADES NERVE EN REDES DE LA UNIVERSIDAD DE CHILE

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

TOMÁS ANDRÉS TORRES BARDAVID

PROFESOR GUÍA:
JOSÉ MIGUEL PIQUER GARDNER

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
EDUARDO RIVEROS ROCA

SANTIAGO DE CHILE
2023

Resumen

La red de la universidad es un recurso indispensable tanto para académicos como estudiantes de la universidad. Y al igual que todas las redes del mundo moderno es susceptible a ataques de individuos u organizaciones maliciosas. Para lidiar con estas y otras amenazas la universidad dispone de la Oficina de Seguridad de la Información(OSI) encargada de proteger la red y mantener una ambiente digital seguro. Entre las labores que realiza esta organización se encuentra la búsqueda de vulnerabilidades en servicios y dispositivos para prevenir la explotación de estos.

Sin embargo, el proceso de ejecución de las pruebas previo a este trabajo sufre de algunos problemas provenientes del hecho de que las pruebas se efectúan manualmente. Como consecuencia el proceso es propenso a errores humanos, necesita de una gran inversión de tiempo para completarse y requiere de cierto conocimiento específico para ser llevado a cabo. Es posible solucionar gran parte de estos problemas al implementar una herramienta capaz de automatizar el proceso de ejecución de las pruebas. En consecuencia, el principal objetivo de este trabajo es implementar la herramienta que mejor se acomoda a las necesidades de la OSI en el ambiente de producción provisto.

Entre la gran cantidad de herramientas comerciales *open source* se optó por extender la herramienta preexistente Nerve. El desarrollo se enfocó en modificar los diferentes módulos de la herramienta para automatizar la ejecución de los *scripts* e incorporar funcionalidades complementarias de gran valor para la oficina. Sumado a esto, se transformaron los *scripts* OSI a un formato estándar para posteriormente incorporarlos en las pruebas de la herramienta. Por último, para garantizar tanto la robustez de la herramienta como el correcto funcionamiento de las funcionalidades desarrolladas se utilizó un ambiente de control enfocado en testeo. De este modo, fue posible desarrollar una solución robusta con funcionalidades y características que permiten abordar las necesidades de la oficina.

En última instancia, se efectuaron pruebas en 5 redes de diferentes organizaciones de la comunidad universitaria. Los resultados obtenidos permiten determinar correctamente los servicios disponibles en las redes y encontrar configuraciones potencialmente vulnerables. Durante la ejecución de la pruebas no hubieron problemas con la herramienta ni grandes impactos en los servicios atacados. En consecuencia, la herramienta desarrollada es capaz de abordar los problemas de la OSI y cumplir con los requerimientos establecidos durante este trabajo.

Agradecimientos

Quiero comenzar esta memoria agradeciendo a mi familia por su soporte incondicional durante todo el período universitario y por la ayuda que me brindaron durante todos estos años.

En segundo lugar, agradecer a mi grupo de amigos del colegio por su constante apoyo durante todo el proceso y lograr levantarme los ánimos en los momentos más difíciles.

También agradecer a mis amigos de la universidad por todas las dificultades y buenos momentos por los que pasamos durante nuestro trayecto universitario.

Además, agradecer al profesor José Piquer por acompañarme a lo largo de este trabajo entregando sus consejos siempre con buena disposición.

Por último, agradecer al personal de la OSI involucrado en esta memoria por su reiterada ayuda desde el inicio del proyecto. En particular a Jaime Fuentes quien fue un gran aporte en las diferentes etapas del trabajo y en quien siempre me pude apoyar cuando surgían inconvenientes o necesitaba orientación.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Hoy en día los ciberataques son cada vez más comunes a lo largo del mundo. Diferentes eventos del ámbito global han contribuido a un gran incremento del cibercrimen en los últimos años. De acuerdo a un reporte de ciberseguridad realizado por Entel Ocean, durante el 2022 el costo global de los ciberataques alcanza aproximadamente los 7.700 millones de dólares al año [1]. Y puesto que las amenazas y riesgos solo siguen aumentando se esperan cifras similares o peores a futuro.

El mismo informe cita diferentes causas que contribuyen al crecimiento de estas actividades. Por un lado, el conflicto entre Rusia y Ucrania motiva el desarrollo de nuevos *softwares* maliciosos y nuevas técnicas de explotación, luego utilizadas a lo largo del globo. Por otro lado, la pandemia del COVID-19 ayudó a la transformación digital de múltiples organizaciones, pero también trajo consigo un aumentó considerable del cibercrimen. Las nuevas tecnologías y la creciente información almacenada por las organizaciones son blancos llamativos para atacantes maliciosos. Sumado a esto, el boom que ha experimentado el retail *online* en los últimos años también favorece al auge de los ciberataques. Por lo que el escenario a nivel mundial incentiva, de cierto modo, este tipo de actividades ilícitas.

La universidad no se ve exenta de estos riesgos, ya que posee una gran cantidad de activos informáticos dedicados a la investigación, desarrollo de actividades educativas y otros servicios. En consecuencia, existe un área especial dentro de la Vicerrectoría de Tecnologías de la Información(VTI) encargada de combatir estos ataques, llamada la Oficina de Seguridad de la Información(OSI). Esta es responsable de desplegar los sistemas de seguridad que permiten a la comunidad universitaria desenvolverse en un ambiente digital seguro, íntegro y confiable.

Para lograr estos desafíos la oficina realiza varias labores como:

- Concientizar a toda la comunidad universitaria en las buenas prácticas y seguridad de la información.
- Verificar cumplimiento de normas internas.
- Análisis permanente de las vulnerabilidades y amenazas a la seguridad de la información.
- Elaboración de matrices de riesgo y planes de mitigación respectivos.

1.1. Motivación

Este trabajo en particular se centra en mejorar el análisis y prevención de vulnerabilidades. De acuerdo al último informe anual de gestión del equipo de respuesta ante incidentes de seguridad informática del gobierno de Chile(CSIRT), algunos de los ataques más connotados del 2022 en el ecosistema nacional se originan en vulnerabilidades que no fueron atendidas oportunamente [2]. Por lo tanto, hoy en día, las organizaciones no sólo tienen el desafío de mitigar las nuevas vulnerabilidades, sino que conocer cuáles son aquellas que yacen en sus sistemas que por diferentes razones no han mitigado aún.

Para lidiar con posibles amenazas de seguridad la oficina realiza un monitoreo permanente de la red de la universidad de Chile para detectar cualquier actividad sospechosa. En conjunto a este monitoreo también se ejecutan pruebas de vulnerabilidades a lo largo de toda la red. Estas pruebas buscan encontrar recursos potencialmente vulnerables antes que otros atacantes. De este modo, se busca evitar que estos recursos puedan ser explotados.

Las pruebas de vulnerabilidades se efectúan utilizando *scripts*, estos ejecutan una secuencia de comandos para detectar o explotar una vulnerabilidad. Las vulnerabilidades pueden afectar diferentes recursos como bases de datos, páginas webs, aplicaciones, etc. En cuanto a los *scripts* utilizados para las pruebas, estos son desarrollados por personal de la OSI o sacados de fuentes externas.

La ejecución de las pruebas es efectuada por personal de la OSI de forma totalmente manual. Este proceso también requiere de acciones complementarias, como la necesidad de interpretar los diferentes *outputs*, la elaboración de reportes y el envío de estos a los encargados de los sistemas. Sumado a esto, se requiere cierto grado de conocimiento específico y manejo de comandos de consola para efectuar las pruebas. Por consiguiente, la barrera de entrada para completar el proceso es bastante significativa. Y también es necesario destinar una cantidad considerable de tiempo para completar todas las tareas.

1.2. Objetivos

1.2.1. Objetivo general

El principal objetivo del presente trabajo de memoria es introducir una herramienta capaz de automatizar las tareas antes mencionadas en el ambiente de producción de la OSI. De este modo, se busca aliviar la carga de los operadores de la herramienta y facilitar el proceso de ejecución de *scripts*. Posteriormente se espera que la oficina adopte la herramienta para la búsqueda de vulnerabilidades en la red de la universidad.

Para alcanzar el objetivo principal el trabajo se centra en encontrar una herramienta existente, extenderla para adecuarse a las necesidades de la oficina e incorporarla al ambiente de producción.

1.2.2. Objetivos específicos

Los objetivos específicos que se derivan del objetivo general son los siguientes:

- Investigar y evaluar diferentes herramientas.
- Seleccionar la herramienta que mejor se acomoda a la situación de la oficina.
- Extender la herramienta para automatizar el proceso de ejecución de *scripts*.
- Transformar e incorporar *scripts* OSI a la herramienta.
- Desarrollar funcionalidades necesarias complementarias al proceso de ejecución de *scripts*.
- Instalar la herramienta en ambiente de producción de la OSI.

1.2.3. Evaluación

Por último, también se busca realizar pruebas para validar diferentes aspectos de la solución. En particular, la evaluación de la solución se aborda desde 2 principales aristas:

- Para verificar la robustez y correcto funcionamiento de la solución se dispondrá de un ambiente de control enfocado en la búsqueda de errores. El ambiente se utiliza para testear los *scripts* y funcionalidades desarrolladas.
- Sumado a esto, se efectuaran pruebas de vulnerabilidades en máquinas reales de la red de la universidad. De forma de poder evaluar el desempeño de la herramienta en un ambiente real y verificar su utilidad en este contexto.

1.3. Herramientas analizadas

Existen herramientas que permiten abordar las problemáticas planteadas en este trabajo llamadas *Escáners de Vulnerabilidades*. Dada la gran cantidad de herramienta comerciales de este tipo ya disponibles, es que este trabajo se enfoca en extender una de la herramientas preexistentes de acuerdo a las necesidades de la oficina. Posteriormente se busca introducir la solución en el ambiente de producción de la OSI.

Un *Escáner de Vulnerabilidades* permite monitorear o atacar diferentes servicios para encontrar fallas de seguridad. Se apoya en *scripts* para efectuar pruebas y encontrar recursos potencialmente vulnerables. Muchas de estas herramientas poseen elementos complementarios que le brindan beneficios a los usuarios, como una interfaz gráfica para utilizar la herramienta, un panel de control para visualizar los resultados obtenidos, diferentes modos de ataque, entre otros.

Dependiendo de los objetivos de las pruebas y la forma de ejecución de estas, las herramientas pueden clasificarse en diferentes categorías. En este trabajo en específico los ataques apuntan a servicios vulnerables presentes en la red de la universidad. Y los escáners que realizan estas pruebas caen en la categoría de *Escáners de Redes*.

Los escáners también se diferencian por algunas funcionalidades específicas que los distinguen. Estas varían desde el método utilizado para efectuar las pruebas hasta el impacto de estas pruebas sobre sus objetivos. Particularmente para este trabajo es de gran

importancia que la herramienta presente la característica *open source*. Dado que esta permite modificar la solución de acuerdo a las necesidades de la oficina.

Las herramientas que mejor se adecúan a las condiciones del trabajo corresponden a Nerve y a Greenbone Vulnerability Manager(GVM) [3, 4]. GVM tiene la ventaja de abordar una gran gama de vulnerabilidades y poseer una mayor cantidad de funcionalidades. Sin embargo, la facilidad para extender y operar la herramienta Nerve hacen que esta sea preferible sobre GVM.

Esta decisión determina en gran medida los problemas a abordar durante el desarrollo de la solución. Uno de los principales desafíos es la incorporación de los *scripts* OSI en la herramienta Nerve. Esto último permite personalizar las pruebas efectuados y encontrar vulnerabilidades específicas. Sin embargo, la herramienta no es capaz de ejecutar los diferentes tipos de *scripts* OSI, por ende se opta por transformar los *scripts* al formato *Nmap Scripting Engine*(NSE) [5]. Este formato no es soportado nativamente por la herramienta, pero gracias a la característica *open source* de esta es posible extenderla para que soporte el formato en cuestión. Al finalizar, basta con transformar los *scripts* al formato NSE e integrarlos a la herramienta para utilizarlos en las pruebas de vulnerabilidades.

1.4. Desarrollo e implementación de la solución

A pesar de seleccionar la herramienta que mejor se acomoda a la situación de la oficina, aún falta incluir nuevas funcionalidades en la solución para abordar todos los requisitos del trabajo. Este proceso también involucra mejorar algunas funcionalidades preexistentes y expandir otras tareas relacionadas a la ejecución de *scripts*. Durante esta etapa se abordan temas como la automatización del proceso de ejecución de *scripts*, la transformación de *scripts* al formato NSE e implementación de una opción para agendar pruebas de vulnerabilidades. El desarrollo de estas funcionalidades requirió entender el funcionamiento de los módulos internos de la herramienta, aprender el lenguaje de programación Lua y manejar herramientas externas como Nmap [6, 7].

En paralelo al desarrollo de los requerimientos se realizan pruebas para verificar su correcta implementación. Las pruebas se llevan a cabo en un ambiente de control local con propiedades similares al ambiente en donde se busca introducir la solución. De este modo, es posible arreglar fallas antes de pasar la herramienta a producción y así, garantizar cierto grado de robustez en la solución final.

La instalación de la herramienta se realiza en una máquina remota de la oficina. El proceso involucra instalar el *software* con sus dependencias y configuraciones de acuerdo a las limitaciones establecidas por la oficina. Posterior a la instalación se efectúan pruebas en 3 sistemas de control para garantizar el correcto funcionamiento de la herramienta.

1.5. Metodología y estructura del documento

Para lograr los objetivos del trabajo se adoptó una metodología que se divide en 5 etapas. Cada etapa es indispensable para llegar a la solución final dado que las etapas

posteriores del trabajo dependen de etapas previas. El presente documento se encuentra estructurado de acuerdo a estas etapas. Cada capítulo del documento encapsula la información de una etapa en particular, y el orden de los capítulos se encuentra directamente ligado a la secuencia de etapas seguida durante el proceso de desarrollo de la solución. A continuación, se explica a grandes rasgos cada una de estas etapas mencionando sus respectivos capítulos del documento.

En el Capítulo 2 se introducen las condiciones y problemas que motivan el inicio del trabajo, además se menciona la solución propuesta que permite abordarlos. También, se incluye información de todos los aspectos de la solución al igual que información detallada de los requisitos del trabajo. En síntesis, detalla el problema a abordar y específica los requisitos de la solución a construir.

El próximo capítulo, correspondiente al Capítulo 3, presenta y discute la literatura de las herramientas relevantes al trabajo, esto involucra otros aspectos adyacentes a la herramienta que también son de gran relevancia. A fines del capítulo se evalúan las mejores opciones dado el contexto de la solución y se selecciona la que mejor se acomoda a los estándares definidos. De esta forma, se da una visión completa de las potenciales herramientas y se justifica la decisión tomada.

Por su parte el Capítulo 4 aborda toda la etapa de desarrollo del trabajo. Esto abarca principalmente la implementación de las distintas funcionalidades que componen la herramienta y otros requisitos necesarios para cumplir los objetivos definidos. Por consiguiente, el capítulo se enfoca principalmente en los aspectos más técnicos de la solución, justificando las decisiones de diseño y los recursos utilizados.

A continuación, el capítulo Capítulo 5 explica el proceso de incorporar la solución en el ambiente de producción. Se detallan todas las dificultades del proceso, cambios realizados y los resultados finales al probar la herramienta. El enfoque principal del capítulo es la instalación de la herramienta y las pruebas realizados.

Finalmente, en el Capítulo 6 se presentan las conclusiones del trabajo y se ofrecen perspectivas sobre el trabajo futuro. Incluye un análisis de los resultados y reflexiones de los distintos aspectos de la solución. Además, presenta un breve resumen del trabajo con una evaluación más crítica.

Capítulo 2

Problema

2.1. Situación inicial

Entre las labores que desempeña la OSI se encuentra el monitoreo constante de la red de la universidad. Esta actividad contempla la ejecución de pruebas de vulnerabilidades, las cuales permiten encontrar recursos con riesgos de seguridad en la red. La intención de las pruebas es detectar los sistemas vulnerables antes que atacantes maliciosos puedan hacerlo. Así es posible tomar precauciones y prevenir una brecha de seguridad. Estos y otros protocolos contribuyen a mantener segura la red de la universidad.

Las pruebas de vulnerabilidades, previas a este trabajo, son llevadas a cabo por personal de la OSI de forma manual. El proceso de ejecución de las pruebas involucra:

- Configurar correctamente los parámetros de los *scripts* de acuerdo al comportamiento deseado.
- Analizar e interpretar resultados de los *scripts*.
- Generar reportes en el caso que se encuentre una vulnerabilidad.
- Enviar reportes a encargados de sistemas vulnerables.

Sin embargo, el proceso sufre de algunas dificultades en su ejecución. En primer lugar, es necesario dedicar una gran cantidad de tiempo y recursos para ejecutar manualmente todo el proceso. Por ello tiende a ser una actividad de mediano impacto para la oficina dentro de todas las operaciones que desempeña. Y aliviar el trabajo asociado a esta tarea ayudaría considerablemente al personal de la OSI.

Sumado a esto, la ejecución de *scripts* requiere de conocimiento específico por parte de los operadores para lograr una ejecución adecuada. El manejo del proceso necesita un buen entendimiento de comandos de consola y de las cualidades de cada *script*. En consecuencia, la barrera de entrada para gestionar el proceso es considerable e impide que nuevos operadores puedan realizar esta actividad fácilmente. Además, la dificultad de este proceso contribuye a que ocurran errores humanos más fácilmente. En suma, el proceso de ejecución de *scripts* no es amigable para nuevos usuario.

2.2. Solución propuesta

Para solucionar los problemas planteados con anterioridad se opta por introducir una herramienta enfocada en la automatización del proceso de ejecución de *scripts*. Se espera que la herramienta sea capaz de llevar a cabo todo el proceso, desde la ejecución de los *scripts* hasta el envío de los reportes a encargados. Todo esto a partir de mínimas configuraciones por parte del usuario operador. De esta forma, es posible reducir considerablemente la carga de los encargados del proceso de ejecución de *scripts*.

La herramienta busca además facilitar el proceso de ejecución de *scripts*. Esto en gran parte, mediante el uso de interfaces que permiten manejar las diferentes etapas del proceso. Y también mediante un *dashboard* para desplegar los resultados al usuario. Estos elementos al ser fáciles de utilizar e interpretar permiten a nuevos operadores familiarizarse rápidamente y a usuarios más experimentados a cometer menos errores. En síntesis, la incorporación de la herramienta facilita en gran medida la ejecución de las pruebas.

Aprovechando la migración al nuevo sistema, la oficina presenta requerimientos que permiten complementar el proceso de ejecución de *scripts*. Similarmente, también se propone abordar otros problemas más específicos para lograr una mejor solución final. En la próxima sección, se introducen a grandes rasgos las diferentes aristas que abarca el trabajo, las soluciones propuestas y las consecuencias.

2.3. Enfoques del trabajo

Introducir una nueva herramienta en el ambiente de producción de la oficina presenta múltiples desafíos en diferentes ámbitos. Por lo tanto, en conjunto con la oficina, se define una serie de requisitos de las principales ideas a abordar durante el trabajo. De este modo, se establecen las necesidades de la oficina, las aristas necesarias a abordar y los límites de cada tema.

Estos requerimientos son claves para la primera etapa del trabajo, correspondiente a seleccionar la herramienta del trabajo. Se opta por utilizar una herramienta preexistente en vez de desarrollar una propia debido a que existen varias opciones comerciales que permiten abordar las problemáticas en este trabajo. Cada una de estas herramientas presenta pros y contras que son necesarios de evaluar, teniendo en cuenta el contexto del trabajo. De ahí que, es de suma importancia tener claro desde un comienzo los requerimientos de la solución final para seleccionar la herramienta que mejor se acomoda a la situación de la oficina. Estos requisitos pueden agruparse de acuerdo a 5 principales categorías, las cuales se mencionan a continuación:

- Automatización: Requisitos específicos de la automatización del proceso de ejecución de *scripts*.
- Facilidad de uso: Requerimientos relacionados con la usabilidad de la herramienta.
- Incorporación de *scripts*: Requisitos asociados a la incorporación de nuevos *scripts* en la herramienta.

- Complementarios: Abordan diferentes aspectos de la herramienta y del proceso de ejecución de *scripts*.
- Entrega de la solución: Apuntan a introducir satisfactoriamente la solución en el ambiente de la OSI con su documentación asociada.

Una lista completa de los requerimientos se encuentra en la tabla A.1 del anexo. La columna "Ya viene implementado" se utiliza en una sección posterior del documento y puede ser ignorada por ahora. También cabe destacar que dada la gran cantidad de requisitos, en las próximas sub-secciones solo se explicarán las categorías a las cuales estos pertenecen, sin detenerse en los requerimientos específicos.

2.3.1. Automatización

Partiendo por los requisitos enfocados en automatizar el proceso de ejecución de *scripts*, son necesarias varias funcionalidades para alcanzar el objetivo principal. En primer lugar, la herramienta debe ser capaz de ejecutar los *scripts* de la oficina por su cuenta a partir de las configuraciones entregadas por el usuario. Luego debe procesar e interpretar la información obtenida de la pruebas para poder desplegar los resultados finales al usuario. Sumado a esto, si el usuario lo solicita, la herramienta debe generar reportes de los resultados e incluir una opción para enviar los reportes a los encargados de los sistemas. Con todo esto resuelto, se cubre la infraestructura interna mínima necesaria para automatizar el proceso.

Por otro lado, también es requerido que la herramienta incluya múltiples interfaces que permitan manejar el proceso. Estas interfaces deben permitir configurar las pruebas de vulnerabilidades, analizar en detalle los resultados obtenidos de las pruebas y permitir generar los reportes. Así es posible controlar todos los aspectos de la ejecución de *scripts* mediante interfaces gráficas.

2.3.2. Incorporación de *scripts*

Aparte de las funcionalidades antes mencionadas también es necesario incorporar los *scripts* utilizados por la OSI en la herramienta. Durante este trabajo se busca facilitar la introducción de *scripts* OSI a la herramienta y también otros que vayan surgiendo a futuro. Sin embargo, el proceso para añadir nuevos *scripts* no es tan simple debido a que los *scripts* de la oficina no se encuentran escritos en un formato estándar, y no es factible utilizar una herramienta que soporte todo los tipos de formatos debido a la complejidad que esto involucra. En consecuencia, se acordó con la oficina transformar los *scripts* a un solo formato y utilizar una herramienta que soporte el formato definido. A continuación, se mencionan diferentes aspectos a considerar durante el proceso de incorporación de *scripts*.

En primer lugar, es necesario definir el formato al cual transformar los *scripts*. Por lo tanto, la primera etapa contempla evaluar diferentes formatos y escoger el que mejor se adecúa a la situación de la oficina. Bajo este contexto, se espera que el formato escogido cumpla con las 3 siguientes características.

Primero, que sea fácil de desarrollar código. Puesto que esto facilita el proceso de creación de *scripts*, lo que a su vez permite reducir substancialmente el tiempo de desarrollo de estos. De este modo, la OSI puede utilizar el tiempo ahorrado para dedicarse a otras

actividades.

En segundo lugar, se busca que el formato definido pueda incluir opciones, librerías o herramientas complementarias que permitan desarrollar todo tipo de *scripts*. De esta forma, es posible desarrollar *scripts* con diferentes enfoques que pueden ser utilizados en distintos escenarios. Así, independiente del problema que se pretenda abordar es posible desarrollar un *script* adecuado a la situación. A raíz de esto, es que el formato debe proveer la suficiente flexibilidad tal que permita desarrollar diferentes tipos de *scripts*.

La tercera característica se encuentra fuertemente ligada a la comunidad que utiliza el formato, en particular, se busca que el formato seleccionado tenga una gran cantidad de *scripts* preexistentes disponibles para su uso público. Este aspecto es relevante debido a que muchas veces los operadores optan por incorporar *scripts* públicos en sus pruebas. Por consiguiente, seleccionar un formato popular con una comunidad grande y activa suele traducirse en una mayor cantidad de *scripts* disponibles para elegir, lo que permite abordar una mayor gama de vulnerabilidades o seleccionar *scripts* de mejor calidad.

Por lo tanto, primero es necesario evaluar los diferentes formatos disponibles y escoger el que mejor se adecúe de acuerdo a los criterios antes mencionados. Luego, se procede a transformar los *scripts* OSI al formato seleccionado. Sin embargo, desde el punto de vista del cliente, es preferible que el trabajo se centre en desarrollar nuevas funcionalidades e introducir correctamente la herramienta en el ambiente, en lugar de transformar los *scripts*. Puesto que el equipo OSI tiene las capacidades y competencias necesarias para realizar la transformación de los *scripts*, pero no dispone del conocimiento especializado necesario para efectuar las otras tareas mencionadas. De ahí que, se acordó con la OSI que el alcance del trabajo es transformar mínimo 1 *script* al formato y que el resto de los *scripts* serán transformados posteriormente por los especialistas de la oficina. De este modo, no hay necesidad de detenerse mucho es este aspecto y es posible enfocarse en los otros ámbitos del trabajo para brindar una solución final más completa.

El principal requisito asociado a la transformación de *scripts* es que los *scripts* definidos sean robustos frente a diferentes posibles casos. Al momento de ejecutar *scripts* relacionados a la búsqueda de vulnerabilidades hay muchas variables que pueden influir en los resultados obtenidos. Algunas de estas son: credenciales de logeo, tipo de conexión, velocidad de la conexión, entre otras. En consecuencia, se espera que los *scripts* desarrollados tengan en cuenta las distintas posibilidades y se adapten correctamente. De esta forma se eviten errores en los resultados obtenidos y durante el proceso de ejecución, los que pueden ser de gran impacto dependiendo de la situación.

Por último, para efectos del trabajo también es importante que se puedan incorporar nuevos *scripts* a la herramienta fácilmente. Se espera que la oficina adopte nuevos *scripts* en el futuro y es de suma importancia dar opciones para incorporar los *scripts* sin mayores problemas. Además, debe existir la infraestructura necesaria para poder ejecutar los *scripts* con sus parámetros personalizados. Así, es posible mantener la herramienta vigente al largo plazo y ajustar las pruebas a las necesidades de la oficina.

2.3.3. Facilidad de uso

Hay 3 características de la herramienta que contribuyen en gran medida a facilitar su uso. La primera, es que esta incluya opciones que permitan realizar pruebas rápidas o a partir de mínimas configuraciones. De este modo, es posible reducir en gran medida la cantidad de acciones y decisiones que deben realizar los operadores para comenzar las pruebas. Además, sirve para familiarizarse con la herramienta y el proceso de ejecución de *scripts*. En definitiva, es una opción que ayuda a introducir nuevos operadores.

En segundo lugar, se requiere que las interfaces sean fáciles de acceder y navegar, ya que esto afecta en gran medida las capacidades para operar la herramienta. Por último, se espera que las interfaces de la herramientas estén en español. Si bien los especialistas de la OSI se manejan en inglés, al trabajar en español es más fácil adaptarse y entender la información desplegada en su totalidad. En suma, estos 3 requisitos apuntan a mejorar la experiencia del usuario, facilitar el uso de la herramienta y disminuir la complejidad de esta para nuevos operadores.

2.3.4. Requisitos complementarios

El trabajo también contempla otras funcionalidades que no están directamente asociadas a solucionar los principales objetivos del trabajo o la integración de la solución. Estas brindan otros beneficios u abordan otros problemas más específicos que, por lo general, no tienen mucha relación entre sí. En la presente sección se agrupan estos diferentes tipos de requisitos.

Para empezar, se busca que la herramienta incluya opciones de pruebas más especializadas y altamente personalizables. Esto es especialmente valioso cuando es requerido realizar pruebas específicas en ciertos sistemas, ya que permite ajustar las pruebas de acuerdo a las propiedades del objetivo. De este modo, el operador tiene un mayor control sobre la ejecución de los *scripts* y es capaz de abordar un mayor rango de escenarios.

Por otro lado, la oficina valora en gran medida la capacidad de agendar las pruebas de vulnerabilidades. La gran ventaja de esta funcionalidad es que quita la necesidad de estar presente durante la ejecución de las pruebas, por lo que puede ser útil en varias situaciones. En la sub-sección Modos de Escaneo se discute más al respecto.

Para la oficina también es relevante la capacidad de realizar pruebas con autenticación en ciertos casos. Esta funcionalidad permite el uso de credenciales en la ejecución de las pruebas y puede ser útil para analizar servicios particulares. Sin embargo, no es un requisito tan rígido como el resto, puesto que son aceptables otras soluciones que permitan simular este comportamiento de otras maneras. Más información se detalla en la sub-sección Escaneos Autenticados vs No-Autenticados.

Adicionalmente, se espera que la herramienta entregue un indicador de severidad asociado a las vulnerabilidades encontradas. En el área de ciberseguridad muchas veces es necesario priorizar la remediación de una vulnerabilidad por sobre otras, por lo tanto, es de gran importancia un indicador que permita determinar cual vulnerabilidad presenta un mayor riesgo. De esta forma, los administradores de sistemas son capaces de abordar las

vulnerabilidades más críticas por sobre las más inofensivas.

El trabajo además considera otros aspectos no funcionales de la herramienta, en especial, el desempeño de la herramienta sobre los objetivos de la red de la universidad. Puesto que es importante que las actividades relacionadas a la ejecución de *scripts* no impacten en la calidad y desempeño de los servicios atacados. También, se espera que la herramienta sea robusta en los diferentes ámbitos que abarca. De esta forma, se gana cierta confianza y dependencia en el *software*, lo que ayuda a tomar decisiones con más certeza.

2.3.5. Entrega de la solución

El trabajo contempla la instalación de la herramienta en el ambiente de producción de la oficina. Y también considera un plan de testeo para garantizar que todo funcione adecuadamente. Las pruebas además sirven para confirmar que la herramienta es una buena alternativa para encontrar vulnerabilidades. En consecuencia, se acordó con la oficina realizar pruebas de vulnerabilidades en la red de la universidad. De este modo, es posible corroborar la utilidad de la herramienta en un ambiente real y verificar su correcto funcionamiento.

Aparte de la instalación, configuración y testeo, el trabajo también abarca la persistencia, mantención y uso de la herramienta una vez finalizado el trabajo de memoria. Para abordar estos últimos puntos se busca capacitar al personal de la oficina en la instalación de la herramienta, navegación de interfaces, agendado de pruebas, entre otros. Así, se espera que el usuario pueda operar la herramienta adecuadamente y mantenerla a futuro.

De forma complementaria, se dispondrá de documentos con información detallada de la herramienta. La documentación debe ser suficiente para poder manejarla en su totalidad. Esto abarca los pasos de instalación, el detalle del método para agregar nuevos *scripts*, como manejar la herramienta y otra información útil para desarrolladores. En definitiva, se busca que la documentación responda gran parte de las preguntas que puedan surgir respecto a la herramienta y que permita el uso adecuado de esta.

A modo de resumen, al finalizar el trabajo la solución final debe cumplir con todas las funcionalidades mencionadas a lo largo de esta sección, quedar instalada correctamente en el ambiente de producción, ser capaz de ejecutar los *scripts* de la oficina e incluir la documentación correspondiente.

Capítulo 3

Escáners de vulnerabilidades

Para resolver los problemas planteados con anterioridad se utilizarán herramientas ya existentes llamadas *escáners de vulnerabilidades*. Estas herramientas se encargan de realizar pruebas de vulnerabilidades en distintos sistemas en base a configuraciones definidas por un usuario. Las pruebas también son conocidas como *escaneos*. Además brindan diferentes funcionalidades complementarias que permiten mejorar el proceso de ejecución de *scripts* en varios ámbitos como rapidez, comodidad, robustez, etc.

La principal ventaja de utilizar estos escáners es que permiten automatizar el trabajo que normalmente debería realizar manualmente la OSI. Este tipo de herramientas son altamente configurables para permitir la ejecución de pruebas de vulnerabilidades sin la necesidad de intervención humana. También poseen funcionalidades que facilitan su usabilidad, como una interfaz para interactuar con la herramienta y opciones para escaneos rápidos. En general, un escáner de vulnerabilidades es una opción útil, eficiente y cómoda para resolver los problemas de la OSI.

La mayoría de las organizaciones requiere una mezcla de escáners para proteger todos sus activos informáticos. Por lo general, cada escáner se enfoca en un tipo de servicio o método en específico. De ahí que, para proteger todas las máquinas de una organización al 100 % es importante utilizar múltiples escáners con diferentes enfoques.

Sin embargo, muchas veces es difícil compatibilizar todas las herramientas en un único punto de almacenamiento de información. Además, esta configuración requiere de una mayor cantidad de personal para manejar cada herramienta. Esto hace que no sea una alternativa viable para un gran número de organizaciones pequeñas o medianas.

La oficina por su parte no presenta ningún tipo de escáner en su ambiente previo a este trabajo. Tampoco dispone de mucho personal como para poder manejar una configuración de múltiples escáners. Por lo tanto, se opta por una configuración simple basada en un escáner principal. Idealmente a futuro iterar sobre el escáner para cubrir vulnerabilidades más específicas o incorporar otros escáners más especializados para brindar una mayor cobertura. Pero de momento con el personal limitado de la oficina y sin la necesidad de realizar pruebas tan especializadas basta con un escáner principal más general.

Cabe recordar que este trabajo considera el uso de una herramienta preexistente para

satisfacer las necesidades de la oficina. De esta forma, no es necesario iniciar un desarrollo desde cero y es posible lograr una solución más completa al iterar sobre una herramienta base. En consecuencia, la primera etapa del trabajo contempla la búsqueda de la herramienta que mejor se adecué al problema planteado. En el presente capítulo se pretende analizar las diferentes alternativas relevantes con sus respectivas fortalezas y debilidades. Luego, se explicarán en detalle las justificaciones y criterios aplicados para tomar la decisión final. Por último, se hablará del formato de *script* seleccionado para este trabajo con las justificaciones correspondientes.

3.1. Tipos de escáners

El primer paso para encontrar la herramienta adecuada corresponde a filtrar los escáners de Vulnerabilidades con enfoques distintos a los buscados en este trabajo. A continuación, se evalúan los 5 principales tipos de escáners de Vulnerabilidades de acuerdo a su relevancia con este proyecto [8, 9]. En caso que un tipo de escáner no sea adecuado en el contexto del trabajo, se descartará. De este modo es posible filtrar a grandes rasgos las potenciales herramientas.

3.1.1. Escáners basados en el *host*

Este tipo de escáners se caracteriza por su forma de uso. A diferencia del resto de los escáners que realizan ataques en función de direcciones IP y puertos externos, este se instala directamente en el sistema que se pretende atacar. De este modo, brinda una perspectiva diferente de las vulnerabilidades presentes en los sistemas y dispositivos a analizar. Su principal ventaja es que permite obtener información que normalmente es muy complicada o imposible de obtener mediante ataques externos. Y dependiendo de la vulnerabilidad puede ser capaz de remediar fallas y prevenir posibles ataques.

Entre sus facultades se encuentra la capacidad de analizar las aplicaciones presentes en una máquina, las configuraciones del sistema operativo y las versiones de los parches. En el caso de detectar una configuración o parche con una potencial falla se envía una notificación al encargado. En algunos casos, incluye también una opción para solucionar automáticamente las fallas detectadas. Adicionalmente, varias herramientas incluyen integración con sistemas de monitoreo de red, de ahí que es una opción conveniente si también es necesario este servicio.

Sin embargo, este enfoque en el contexto del trabajo sufre de una gran desventaja, correspondiente a la necesidad de instalar la herramienta en el sistema a escanear. Esto no es viable cuando es requerido analizar todos los servicios de la red de la universidad, ya que en muchos casos la oficina no tiene acceso a ellos directamente o carece de los permisos necesarios. Lamentablemente muchos de los escáners más modernos y utilizados caen en esta categoría, en consecuencia, debieron ser descartados.

3.1.2. Escáners de aplicaciones Web

Los escáners de aplicaciones web se diferencian del resto por su enfoque único en aplicaciones web. Brindan funcionalidades especializadas en la búsqueda de

vulnerabilidades en interfaces web y fallas en el *backend* de la aplicación. Lo que permite a muchas organizaciones verificar que sus aplicaciones sean seguras.

Hoy en día, las páginas web pueden llegar a ser altamente complejas dependiendo de las funcionalidades que esta proporcione. No es raro encontrar múltiples lenguajes de programación, librerías y módulos coexistiendo dentro de estos sistemas. Esta complejidad trae a sus vez una mayor posibilidad de presentar vulnerabilidades pero dificulta la capacidad para encontrarlas. Por consiguiente, muchas veces no basta con herramientas comunes y es necesario una herramienta especializada en este tipo de servicios para detectar las vulnerabilidades y remediarlas.

En cuanto a este trabajo, no basta con un análisis de las páginas webs disponibles para asegurar la integridad de la red. Más bien es necesario un escáner con un enfoque más general que pueda abordar múltiples tipos de servicios sin la necesidad de tanto detalle. Logrando así una una mayor cobertura de los servicios de la red. A raíz de esto es que también se descarta esta opción.

3.1.3. Escáners de bases de datos

Este tipo de escáners se especializa en la búsqueda de potenciales vulnerabilidades en servicios de bases de datos. A pesar de que su enfoque es distinto al de los otros tipos de escáners, este también sufre de los mismos problemas que los escáners de aplicaciones web. Particularmente, los servicios objetivos de los ataques son muy específico y poco útiles en el contexto del trabajo. En consecuencia, los escáner de este tipo también fueron descartados tempranamente.

3.1.4. Escáners de vulnerabilidades en la Nube

Corresponden a herramientas diseñadas para analizar el ambiente de la nube de una organización. En cierto aspecto se parecen a los escáners basados en *hosts*, ya que se instalan en la nube del proveedor y se encargan de buscar malas configuraciones en los sistemas, formas de acceso no permitidas y monitoreo de datos. Sin embargo, a diferencia de los escáners basados en *hosts* generalmente no tienen acceso a todos los recursos disponibles debido a que el proveedor de la nube mantiene ciertas restricciones. En consecuencia, la seguridad de los sistemas corresponde a una responsabilidad compartida entre el proveedor y el usuario que utiliza los sistemas.

En el caso de la red de la universidad, esta se encuentra principalmente fuera de la nube por lo que no es una alternativa factible para la gran mayoría de los sistemas que se piensa escanear. Eventualmente, si se migran más sistemas a la nube podría ser una opción válida pero está lejos de ser el principal foco de vulnerabilidades.

3.1.5. Escáners de redes

Los escáners de redes son las principales herramientas utilizadas para detectar riesgos en redes. En primera instancia, realizan un reconocimiento de los servicios y dispositivos presentes en la red. Posteriormente, ejecutan escaneos en los computadores y sistemas detectados para encontrar vulnerabilidades. En particular, utilizan *scripts* provenientes de diferentes bases de datos para atacar todo tipo de vulnerabilidades dependiendo del tipo de

objetivo. De esta forma tratan de simular los ataques que realizan *hackers* externos para entender los puntos débiles de la red.

La mayoría de estas herramientas permite detectar vulnerabilidades en múltiples tipos de servicios y protocolos como SSH, FTP, Interfaces Web, Bases de Datos, entre otros. Al abarcar una gran gama de vulnerabilidades, las pruebas tienden a ser de carácter más general en contraste con otras herramientas especializadas en un tipo de servicio en particular. Pero gracias a esto último, también es posible abordar una mayor cantidad de vulnerabilidades. Esto se alinea en gran medida con las características de la herramienta buscada por la OSI.

3.1.6. Conclusión

La oficina al depender de un único escáner de vulnerabilidades para abordar toda la red de la universidad necesita de una herramienta capaz de abarcar una gran cantidad de servicios y dispositivos. Bajo esta restricción solo los escáners basados en el *host* y los escáners de redes son viables, el resto tienden a enfocarse en un solo tipo de servicio y no brindan la suficiente cobertura a toda la red. El primer tipo tampoco es viable en el contexto del trabajo debido a la falta de permisos y restricciones de accesibilidad que limitan el uso de este tipo de herramientas en los computadores de la universidad. Los escáners de red por su parte no sufren de estos problemas y poseen la ventaja de que utilizan *scripts* para efectuar sus pruebas. Por lo tanto, dadas las condiciones del trabajo y la necesidad de incorporar *scripts* OSI a la herramienta, es que estos escáners son los que mejor se acomodan a los requerimientos del trabajo.

3.2. Características escáners de vulnerabilidades

Aparte de las distintas categorías mencionadas previamente, los escáners de vulnerabilidades además poseen características distintivas que permiten diferenciar la forma en que estos realizan escaneos. Estas características brindan ventajas y desventajas dependiendo del enfoque que se le de a la herramienta. Por lo general, estas no son intrínsecas de un tipo de escáner en particular, si no más bien pueden presentarse en cualquiera de los tipos mencionados con anterioridad, ya que afectan la forma de operar de los escáners. A continuación se mencionan algunas de las características más relevantes de los escáners y su relación con este trabajo [10, 11].

3.2.1. Escaneos internos vs externos

Los escaneos internos se realizan desde el interior de la red. Estos asumen que un atacante ya tiene acceso a la red interna de la organización y planea explotar vulnerabilidades mediante *escalación de privilegios* y/o abuso de permisos para causar daños. El término *escalación de privilegios* se refiere al acto de ganar acceso a recursos que normalmente están protegidos para el usuario mediante diferentes técnicas de explotación de *software* [12]. Generalmente apuntan a tomar control del dispositivo o servicio atacado, por lo cual pueden tener grandes impactos dependiendo del objetivo.

En contraste, las escaneos externos se realizan sin los permisos para acceder a la red

interna. Apuntan a explotar vulnerabilidades disponibles en servicios públicos como puertos abiertos, páginas webs y protocolos. Las condiciones necesarias para efectuar este tipo de ataques son menos estrictas que el otro tipo de pruebas. Pero en la mayoría de los casos el impacto de las vulnerabilidades también es menor. La mayor diferencia con el otro tipo de escaneo se encuentra en el método utilizado para detectar las vulnerabilidades. Las pruebas efectuadas en este caso se realizan considerando que el atacante no tiene conocimiento o acceso de recursos internos y solo puede explotar recursos públicos.

Este trabajo apunta a la detección de vulnerabilidades dentro de la red de la universidad. Idealmente se busca cubrir tanto vulnerabilidades internas como externas de la red, sin embargo, muchas veces no es posible realizar pruebas tan robustas como para cubrir ambos aspectos. Por lo general, los escáner de vulnerabilidades de redes se centran principalmente en la ejecución de escaneos externos, ya que sus ataques involucran el uso de puertos abiertos en la máquinas. Por lo que este trabajo se centrará en escaneos de este tipo.

3.2.2. Escaneos autenticados vs no-autenticados

Otra opción relevante a la hora de llevar a cabo un escaneo corresponde al tipo de autenticación. Muchas páginas o servicios necesitan de ciertas credenciales para acceder a ellos. Por lo general, los servicios que requieren de accesos con privilegios tienden a contener una mayor cantidad de información sensible o de importancia para la organización. En consecuencia, las vulnerabilidades que presentan estos sistemas tienden a ser más críticas que en sistemas sin autenticación. Hoy en día, casi todas las herramientas realizan escaneos sin el uso de credenciales pero cada vez son más las que incluyen configuraciones que permiten autenticarse en los diferentes sistemas de logeo.

Esta configuración no es necesaria en el contexto de este trabajo, pero dependiendo del uso que se le de a la herramienta puede ser una característica de gran valor para la oficina. De ahí que, una herramienta con funcionalidades que permitan la autenticación en diferentes sistemas tiene un valor agregado comparada con otras herramientas que no incluyen esta opción. Sin embargo, es posible simular en parte esta característica utilizando los parámetros de los *scripts*. Muchos de estos poseen parámetros con credenciales de logeo que pueden ser utilizados para realizar ataques autenticados. De este modo, el operador de la herramienta puede efectuar escaneos autenticados al utilizar los parámetros adecuados durante las pruebas.

3.2.3. Escaneos intrusivos vs no-intrusivos

También es importante estudiar el método utilizado para atacar los objetivos. Existen 2 principales formas de realizar los ataques, correspondientes a escaneos intrusivos y no-intrusivos. Los primeros apuntan a explotar directamente las vulnerabilidades presentes en los sistemas. Por lo general, dependiendo del impacto que generan los pruebas se pueden clasificar según distintos niveles de intrusividad. Por otro lado, se encuentran los no-intrusivos, especializados en inspeccionar y revisar los sistemas objetivos más que explotar las vulnerabilidades directamente. A partir de estos, se obtiene información de los atributos y posibles problemas presentes en los sistemas. De esta forma, los escaneos de este tipo son capaces de indicar potenciales vulnerabilidades con cierta confianza sin efectuar el ataque propiamente tal, y así evitar cualquier impacto sustancial en los sistemas objetivos.

En el contexto del trabajo, se busca que los escaneos no impacten los servicios o sistemas que son escaneados. Pero a su vez, no basta con inspeccionar los elementos de la red de la universidad, si no que es necesario un análisis más detallado de las vulnerabilidades presentes. De ahí que, es necesario efectuar escaneos con bajo grado de intrusividad tal que no impacten considerablemente los sistemas o servicios escaneados pero brinden información adicional. Por lo tanto, a la hora de escoger una herramienta, es preferible una que permita filtrar los *scripts* de acuerdo a su nivel de intrusividad, de modo que sea fácil ejecutar *scripts* de bajo impacto.

3.2.4. Modos de escaneo

Otra arista a considerar son los modos de escaneo. Dependiendo de las necesidades de los usuarios puede que no baste con la capacidad de realizar un escaneo en un determinado momento. En sistemas críticos, es recomendado realizar escaneos periódicos o continuos dependiendo del enfoque. Un escaneo continuo como su nombre lo indica es el que constantemente se está realizando en el objetivo determinado. De esta manera identificando vulnerabilidades lo más temprano posible pero a costas de cierto impacto en el desempeño del sistema.

Por otro lado, están los escaneos con capacidad de ser agendados, la gran ventaja de estos es que permiten adaptarse al horario del usuario. Muchas veces, es necesario realizar los pruebas fuera de los horarios regulares de trabajo o simplemente no hay disponibilidad para supervisar el proceso directamente. En tales casos, es posible configurar el escaneo para realizarse a la hora que se estime conveniente. Reduciendo de esta manera las interacciones y compromiso necesario para lograr un escaneo satisfactorio.

Por último, se encuentran los escaneos periódicos correspondientes a una mezcla de los otros escaneos mencionados previamente. El concepto involucra realizar escaneos cada cierto intervalos continuos en sistemas vulnerables. De esta manera, obteniendo resultados periódicamente de los sistemas objetivos. El Instituto Nacional de Estándares y Tecnología(NIST) recomienda realizar escaneos al menos anualmente en todos los sistemas, lo que puede ser antes dependiendo del riesgo al que estén sometidos [13]. Por lo general, muchas organizaciones realizan estos escaneos 1 vez al mes o cada cuarto del año dependiendo de la complejidad del escaneo [14]. En consecuencia, puede ser de gran utilidad contar con un escáner con funcionalidades que apoyen estas frecuencias de escaneos.

Para la OSI basta que la herramienta tenga una opción para realizar los escaneos inmediatamente y otra para agendar los escaneos. El resto de las funcionalidades mencionadas son convenientes pero no necesarias para el uso que se le va a dar.

3.2.5. Conclusión

A modo de resumen, existen diferentes características a considerar a la hora de seleccionar un escáner de vulnerabilidades para este trabajo. En la lista a continuación se encuentra la información más importante respecto a cada característica en particular. En el caso que una de estas características sea requerida para la solución final esta se incluye en la lista de requisitos del trabajo.

- El método de escaneo (externo o interno) no es de gran importancia para efectos del

trabajo. Por lo general, los escáners de redes tienden a realizar escaneos externos.

- Por otro lado, es requerida una forma realizar escaneos autenticados en la red. En el caso que la herramienta no disponga de tal opción es aceptable pasarle la información directamente a los *scripts* ejecutados mediante parámetros.
- En tercer lugar, es importante que la herramienta permita realizar escaneos intrusivos de bajo impacto. De este modo, no se ve afectado substancialmente el desempeño de los servicios y es posible obtener información más detallada que escaneos superficiales.
- Por último, para la oficina es suficiente que la herramienta incluya una opción efectuar escaneos inmediatamente y otra para agendar *scripts*.

3.3. Herramientas potenciales

3.3.1. Filtrado general

Una de las decisiones más importantes del trabajo corresponde a seleccionar la herramienta que mejor se acomoda a los requerimientos buscados por la OSI. Por lo tanto, se estudiaron una gran cantidad de escáners de vulnerabilidades candidatos con diferentes características y enfoques. El análisis resulto en más de 30 escáners de vulnerabilidades distintos, por lo que fue necesario aplicar un filtrado a gran escala para reducir las potenciales alternativas.

El proceso de filtrado consiste principalmente en verificar que las herramientas cumplan con los requisitos establecidos por la oficina y que sean del tipo escáner de redes. Sin embargo, cabe mencionar que se aplicaron un par de excepciones a las reglas. Primero, a raíz de que la mayoría de las herramientas no se encuentra disponible en español, se permite que la interfaz web de la herramienta seleccionada esté en ingles, dado que el personal de la oficina también se maneja en este idioma. Segundo, en el caso que una herramienta sea del tipo *open source* o pueda ser extendida de alguna manera, esta no requiere cumplir con todos los requisitos del trabajo. Puesto que al poseer la capacidad para ser extendida si la herramienta no cumple con algún requisito se espera que este sea desarrollado o incorporado durante el trabajo. Por lo que si se opta por esta alternativa, basta con extender la herramienta hasta lograr todos los requisitos propuestos por la oficina.

Como resultado de este filtrado a gran escala se obtuvieron 5 herramientas con cualidades prometedoras. El resto de las alternativas fueron descartadas por no cumplir con algunos de los requerimientos. Dada la gran cantidad de herramientas no es factible hablar de las características particulares de cada una de ellas y las razones por las que fueron descartadas. No obstante, sí se mencionarán a continuación los criterios para descartar herramientas más comúnes.

En primer lugar, muchas de las herramientas más modernas y del estado del arte fueron descartadas por ser del tipo *basadas en el host*. Esto ocurre debido a que gran parte de estas herramientas proporcionan servicios de monitoreo continuo de los objetivos, los cuales requieren de la instalación de *agentes* en las máquinas monitoreadas. Un agente corresponde a *software* ligero encargado de recopilar información del sistema en el cual se

instala, posteriormente se envía esta información al núcleo del escáner de vulnerabilidades para tomar decisiones. El problema asociado a este tipo de configuración es que no es factible instalar los agentes en dispositivos que no son manejados por la oficina. Por lo tanto, no es una opción viable para abordar toda la red y, como consecuencia, es necesario descartar gran parte de las herramientas totales.

Otra funcionalidad que no muchas herramientas poseen es la capacidad de incorporar *scripts* a la herramienta. Gran parte de las herramientas hoy en día se basan en grandes bases de datos de vulnerabilidades para ejecutar escaneos, a tal punto que tan solo se permite la ejecución de *scripts* disponibles en estas. Esto puede ser conveniente para muchos administradores de sistemas, ya que les permite estar al tanto de las últimas vulnerabilidades descubiertas sin la necesidad de dedicar tiempo a desarrollar *scripts*. Pero tienen la desventaja de que limitan en gran medida la flexibilidad del usuario operador de la herramienta. En el contexto del trabajo, no es aceptable una herramienta que carezca de está funcionalidad. Idealmente se espera que la herramienta final incluya *scripts* provenientes de bases de datos y también una opción para agregar propios si es necesario.

En las próximas secciones se explorará más en detalle cada una de las distintas alternativas que no fueron descartadas luego del filtrado general. Analizando las principales fortalezas y debilidades de cada una para luego escoger la mejor herramienta de acuerdo al problema.

3.3.2. Tripwire IP360

Tripwire IP360 es un escáner de vulnerabilidades pagado que se caracteriza por analizar todos los dispositivos y servicios en un red mediante ataques eficientes, considerando las características de los objetivos [15]. Permite agendar escaneos y generar varios tipos de reportes, pero carece de una opción nativa para enviar los reportes a encargados. También, permite la ejecución de *scripts* personalizados en la herramienta para varios casos de uso. De este modo, logra abarcar en gran parte las funcionalidades buscadas.

La herramienta destaca por su capacidad para entregar mejores puntajes asociados al nivel de riesgo de una vulnerabilidad. La mayoría de los escáners de vulnerabilidades utiliza etiquetas comunes para clasificar la severidad de las vulnerabilidades. Generalmente estas etiquetas van desde *Bajo* hasta *Crítico*, pero puede que algunas veces sean insuficientes para categorizar las millones de vulnerabilidades existentes. En consecuencia, Tripwire IP360 asigna un valor numérico a las vulnerabilidades para indicar su peligro. Mientras mayor sea el número asignado mayor será el impacto o riesgo asociado. Así logrando una mejor comparación entre vulnerabilidades que las opciones provistas por otras alternativas.

Sumado a esto, la herramienta brinda distintos tipos de reportes dependiendo de la audiencia a la cual se presenta la información. Por ejemplo: la alternativa para ejecutivos se apoya en estadísticas globales y costos asociados. Por otro lado, los reportes enviados al área de seguridad poseen información particular más detallada de las vulnerabilidades y analítica avanzada que la complementan. De esta forma, se logra adecuar a las necesidades del usuario final.

La única desventaja considerable que presenta la herramienta en comparación con las otras es la carencia de una opción para enviar reportes a encargados. Si bien esta limitación

impide completar el proceso de ejecución de pruebas de vulnerabilidades planteado por la OSI, existen alternativas para lograr la funcionalidad usando otros métodos. Por ejemplo, la librería *inotifywait* permite detectar cuando se crea un archivo y configurar acciones sobre este [16]. Por lo tanto, sería posible determinar la generación de reportes por parte de la herramienta y configurar el envío a encargados particulares basados en la dirección IP del reporte. No obstante, pese a que esto permite solucionar el problema a grandes rasgos, sigue siendo bastante inconveniente para los operadores de la herramienta.

3.3.3. Nessus

Nessus es un escáner comercial de vulnerabilidades distribuido por Tenable, Inc [17]. Al igual que otros escáners de red analiza los puertos disponibles en los sistemas objetivos para determinar los servicios y protocolos disponibles. Luego efectúa ataques a los servicios encontrados utilizando bases de datos de vulnerabilidades comunes. La herramienta se encuentra disponible en una versión gratuita con ciertas limitaciones como la cantidad de dispositivos que se pueden analizar, cantidad de configuraciones disponibles y falta de capacidad para configurar reportes. También incluye funcionalidades como agendado de escaneos y generación de reportes.

La herramienta no dispone de una opción para agregar *scripts* directamente a la herramienta. Pero si permite incorporar *plugins*. Estos corresponden a piezas de *software* desarrolladas por la comunidad con distintas funcionalidades extras que se pueden agregar a la herramienta. Así, permitiendo extender la herramienta para lograr ciertas funcionalidades adicionales no disponibles en la versión base. Lo que puede ser utilizado para agregar *scripts* desarollados por la OSI en el formato Nessus Attack Scripting Language(NASL) compatible con Nessus [18].

Sin embargo, existen algunas desventajas que son necesarias a considerar a la hora de desarrollar *software* en NASL. Primero, hay muy poca documentación al respecto de como transformar *scripts* al formato deseado, dificultando en gran medida el proceso. En segundo lugar, la API de Nessus y el lenguaje NASL presentan incoherencias y fragmentaciones que interrumpen o complican el desarrollo asociado, posiblemente debido a la antigüedad de la herramienta/lenguaje. Por estas causas es que el desarrollo de *plugins* en Nessus se encuentra en gran parte deprecado.

3.3.4. Nexpose

Nexpose es un escáner de vulnerabilidades desarrollado por Rapid7 [19]. Dispone de una versión gratis para la comunidad con ciertas limitaciones y una versión pagada que permite realizar escaneos fácilmente en sistemas. Basta con indicar las direcciones IP de los objetivos y seleccionar el grupo de *scripts* para empezar un ataque. Una vez concluido el ataque, se generan reportes a partir de los resultados obtenidos con opción para enviarlos a los encargados. Además dispone de varias opciones de escaneo como agendado y uso de credenciales. En consecuencia, corresponde a una solida opción para resolver los problemas planteados.

La incorporación de *scripts* a la herramienta es posible gracias a que esta permite incorporar chequeos de vulnerabilidades en el formato *Nexpose Vulnerability Check*

Data(VCK). Por lo tanto, basta con transformar los *scripts* de la oficina al formato mencionado para lograr incorporarlos a la herramienta [20].

En lo que respecta al formato *.vck*, este es simple de desarrollar pero consta de algunas limitantes. Por un lado, soporta solo cierta cantidad de protocolos y servicios, que a pesar de abarcar los más populares, sigue siendo una restricción a considerar a la hora de desarrollar *scripts*. Además, corresponde a un lenguaje del tipo *Markup* limitado por sus tags, atributos y clases al momento de programar [21]. Tampoco dispone de funcionalidades de programación más complejas basadas en lógica, como uso de loops *for* y *while*. Principalmente se limita a intercambiar *requests* con un servidor y evaluar el contenido de esta información para encontrar vulnerabilidades.

Por otro lado, la herramienta destaca por su capacidad para formar grupos de *scripts* llamados *templates*. Estos *templates* luego se pueden ejecutar sobre objetivos específicos logrando un escaneo más personalizado de la red. La herramienta ya viene por defecto con algunos *templates* que se pueden modificar de acuerdo a las necesidades del usuario.

La herramienta además posee otras funcionalidades complementarias que apoyan al proceso cómo la asignación de puntajes de 1 a 1000 a las vulnerabilidades, lo cual permite destacar de mejor manera el impacto que pueden tener. También incluye reportes de remediación con instrucciones paso a paso de como solucionar o remediar potenciales riesgos conocidos. Y otras funcionalidades útiles dependiendo del enfoque que se le quiera dar, estas se detallan en la página del producto [22].

3.3.5. Greenbone Vulnerability Manager

Greenbone Vulnerability Manager(GVM), es el escáner de vulnerabilidades de redes *open source* más completo y utilizado [4]. Posee constante mantenimiento por parte del equipo de Greenbone y se rige bajo las licencias GNU GPL-2 y GNU GPL-2+ por lo que se puede copiar, modificar y redistribuir de acuerdo a lo estipulado en las licencias. Incluye opciones para escaneos personalizados, agendar escaneos, generar reportes y enviar reporte a encargados. Esto le permite situarse como una opción bastante completa en el contexto del trabajo.

GVM destaca por la gran comunidad que tiene detrás. Esto se ve reflejado en cosas como la mantención de la herramienta. Puesto que gracias al esfuerzo de la gente y el equipo Greenbone, esta se encuentra generalmente al día con las últimas actualizaciones. Además incluye una cantidad considerable de *scripts* compatibles, ingresados por varios usuarios y también provenientes de diferentes fuentes de información. Esto último, siendo bastante raro para herramientas *open source*.

La característica *open source* permite además modificar la solución para adecuarse a las necesidades de los usuarios finales. Sin embargo, es importante considerar que la herramienta se encuentra escrita principalmente en el lenguaje de bajo nivel C, caracterizado por ser difícil de trabajar y extender. Sumado a esto, la herramienta GVM posee varios módulos diferentes con sus propias complejidades, las cuales son necesarias de aprender si se quieren introducir nuevas funcionalidades. Por lo tanto, a pesar que la herramienta da la opción para modificar el *software* a gusto, la gran complejidad para desarrollar nuevas funcionalidades

acompañado de las restricciones de tiempo del trabajo no hacen muy viable el desarrollo de nuevas funcionalidades.

3.3.6. Nerve

Nerve fue desarrollado bajo la idea de que las herramientas comerciales existentes son buenas, pero, en general; pesadas, costosas y difíciles de extender [3]. La herramienta se especializa en encontrar vulnerabilidades del estilo *low-hanging fruit*. Estas vulnerabilidades son relativamente simples y fáciles de arreglar, generalmente provocadas por negligencias o errores humanos. La herramienta además destacar por su facilidad de uso e instalación. Sumado a esto, incluye opciones para realizar escaneos continuos, generación de reportes y envío a encargados.

Sin embargo, la herramienta presenta claras carencias en comparación con otras. En primer lugar, no incluye un sistema que permita realizar escaneos autenticados, necesario para revisar la integridad de algunos servicios. Tampoco incluye una opción para agendar escaneos. Y, en tercer lugar, no posee una opción nativa para incorporar *scripts*, dificultando la integración de los *scripts* OSI. Pese a estos problemas, la alternativa sigue siendo mejor que una red desprotegida, como es el caso de la red de la universidad previo a este trabajo. Por lo que se mantiene como una opción viable para la oficina.

Nerve tiene la ventaja que es de carácter *open source*, pero a diferencia de GVM no presenta las mismas complicaciones para extender la herramienta. En primer lugar, Nerve se encuentra escrito principalmente en el lenguaje de programación *Python* [23], correspondiente a un lenguaje de alto nivel caracterizado por su facilidad para programar en él y su gran número de librerías que permiten abordar todo tipo de desafíos. Además la herramienta posee una estructura interna más simple en comparación con otras herramientas, lo que facilita la comprensión de sus módulos. Estas características hacen que la herramienta sobresalga por su facilidad para ser extendida.

La herramienta, sin embargo, sufre de ineficiencia en sus pruebas de vulnerabilidades. La principal causa proviene de la ejecución de todos sus *scripts* durante las pruebas. En contraste, el resto de las herramientas al efectuar pruebas de vulnerabilidades realizan un chequeo preliminar de los sistemas antes de iniciar el ataque para determinar los servicios y protocolos presentes en los objetivos. De esta manera, se evita ejecutar *scripts* que no calcen con el tipo del objetivo y solo se ejecutan los que sí son relevantes. Logrando así los mismos resultados que si se ejecutaran todos los *scripts* pero ejecutando solo los que son relevantes.

Por su parte Nerve no realiza ningún tipo de optimización previa al ataque. Por lo que sus ataques ejecutan la totalidad de los *scripts* de la herramienta en todas las direcciones IP ingresadas en la configuración. Pese a esto, no hay un gran impacto en el desempeño general comparado con otros escáners por 2 razones. Por un lado, la ejecución de *scripts* en servicios en donde no corresponde ejecutarlos es generalmente de bajo impacto, gracias a que en la mayoría de los casos al producirse una incompatibilidad se termina la ejecución del *script* inmediatamente. Por lo que esto no afecta en gran medida al desempeño.

En segundo lugar, Nerve no posee una gran cantidad de *scripts* en su repertorio, en total son alrededor de 90. Por lo tanto, el impacto sobre el desempeño es substancial en

comparación con otras de las herramientas mencionadas previamente, las cuales poseen cientos de ellos. Además, Nerve por defecto tiende a utilizar *scripts* y configuraciones de bajo impacto en los sistemas o servicios atacados. Y en ciertos casos si un *script* demora mucho en finalizar se cierra a la fuerza. Todo esto permite reducir las consecuencias en el desempeño de la herramienta a pesar de sus fallas. Aún así, es importante considerar que a medida que se vayan introduciendo nuevos *scripts* a la herramienta este problema solo empeorará. Por el momento, a pesar de sus imperfectos la herramienta no tiene un mal desempeño.

Por último, Nerve sufre del problema de que sus *scripts* y base de datos de *scripts* no son constantemente actualizados. En la mayoría de las herramientas previamente mencionadas las bases de datos de vulnerabilidades son actualizadas de acuerdo a nuevas vulnerabilidades y cambios que vayan surgiendo. Sin embargo, en el caso de Nerve estas actualizaciones no se llevan a cabo comúnmente. Esto en parte se debe a que Nerve no necesita ser actualizado tan seguido como otras herramientas dado su enfoque en vulnerabilidades del estilo *low-hanging fruit*. Estas en su gran mayoría ya se encuentran en la base de datos de la herramienta y no varían mucho en el tiempo, por lo que no es necesario actualizar el repertorio muy seguido.

3.4. Criterios Selección Herramienta

Del total de herramientas analizadas existe un grupo considerable que son viables para efectos de este trabajo. En consecuencia, es necesario explorar más en detalle cada una de ellas para luego seleccionar la que mejor se acomoda al problema. De ahí que, surge la necesidad de definir nuevos parámetros más especializados para filtrar aún más las herramientas candidatas. Además es importante verificar que la herramienta cumpla con las funcionalidades que dice tener y evaluar su comportamiento en la práctica. Por esta razón, es que también se contempla ejercer pruebas de vulnerabilidades en un ambiente controlado utilizando las mejores herramientas candidatas.

Un gran aspecto a considerar al momento de introducir la solución es el costo de utilizar la herramienta. En el caso de este proyecto, la mayoría del costo viene de la adquisición o uso del escáner de vulnerabilidades. Es este aspecto las herramientas las herramientas pueden clasificarse de acuerdo a 3 principales categorías:

- Pagadas.
- Gratuitas.
- *Open source*.

Cada una de estas tiene sus propios pros y contras. Los cuales se discutirán a continuación.

3.4.1. Herramientas pagadas

Corresponden a herramientas donde es necesario pagar una suma inicial para su uso y/o pagos continuos cada cierto tiempo. Por lo general, tienen asociados altos costos pero su precio varía dependiendo de la cantidad de funcionalidades provistas y la cantidad de dispositivos en los que se instalará/usará. Normalmente, presentan la ventaja de que poseen múltiples funcionalidades extras y cuentan con soporte por parte de la empresa que provee

el servicio. Por lo general, la mayoría son del tipo *host-based* pero también hay varios escáners de redes.

En lo que respecta a este trabajo, no existe una gran necesidad de este tipo de herramientas. Debido a que el primer análisis realizado sobre las herramientas candidatas arrojó buenos resultados en herramientas sin costos asociados. Por lo tanto, no hay necesidad de optar por herramientas pagadas cuando existen otras sin costos que permiten abordar los mismos problemas. De ahí que, para efectos del trabajo se descartan las opciones pagadas cómo Tripwire IP360 y Nexpose.

3.4.2. Herramientas gratuitas

Las herramientas gratuitas, como su nombre lo indica, corresponden a herramientas sin costos asociados. A diferencia de las herramientas *open source* su código fuente no es público, en consecuencia, no pueden ser modificadas fácilmente para desempeñar funcionalidades adicionales. Gran parte de estas corresponden a las mismas herramientas pagadas pero con ciertas restricciones que las limitan. Por ejemplo, menor cantidad de direcciones IP que se pueden escanear, menor cantidad de interfaces disponibles y reducción en la cantidad de funcionalidades totales. Son buenas opciones cuando se quiere probar el *software* antes de comprar la versión pagada.

En su gran mayoría, este tipo de herramientas es insuficiente para lo buscado en este trabajo. Las distintas limitaciones que presentan se traducen en grandes problemas que hacen que las herramientas no sean viables en el contexto del trabajo. Del total de herramientas gratuitas analizadas *Nexpose* en su versión comunitaria y *Nessus* son las únicas que se acercan a los requerimientos buscados. A pesar de esto, las limitaciones que presentan como cantidad máxima de direcciones IP a analizar y reducción en el número de opciones de escaneo, no permiten usarlas en el ambiente en el que deben desempeñarse. Puesto que la red de la universidad presenta muchos dispositivos en ella y no sería posible analizarlos en su totalidad al utilizar este tipo de herramientas. En definitiva, ambas opciones no son factibles para la oficina.

3.4.3. Herramientas *open source*

La última categoría corresponde a las herramienta *open source*. Estas poseen la gran ventaja de que son gratuitas y su código fuente es de uso público bajo ciertas licencias. Por lo tanto, es posible modificar las herramientas para acomodarse a los requisitos que busca el usuario. Además cuentan con una comunidad colaborativa que ayuda a mejorar y extender la herramienta en distintos ámbitos.

Una de las grandes desventajas de este tipo de herramientas tiene relación con la baja frecuencia con que son actualizados. Esta falta de actualizaciones, por lo general, se debe a que no es factible mantener la herramienta con actualizaciones constantes bajo el modelo de negocios *open source*, ya que muchas veces este tipo de proyectos no son rentables. Desde un punto de vista de seguridad, no actualizar constantemente la herramienta puede significar una gran cantidad de vulnerabilidad dada la frecuencia con la que surgen nuevas vulnerabilidades.

Pese a esta debilidad, las herramientas *open source* siguen siendo las opciones preferibles para este trabajo. Si bien no incluyen tantas funcionalidades y actualizaciones como las otras herramientas, destacan por su capacidad para incorporar las funcionalidades faltantes. Sumado a esto, poseen la ventaja comparativa de que no es necesario pagar o limitar la herramienta. En consecuencia, se prefieren las opciones *open source*, con GVM y Nerve como sus principales exponentes en el contexto de este trabajo.

3.4.4. Facilidad de extender

Luego de descartar las herramientas no pertenecientes a la categoría *Open Source*, las opciones se reducen a GVM y Nerve. Ambas herramientas presentan claras diferencias pero comparten que son incapaces de satisfacer todos los requisitos buscados por la oficina de forma nativa. Por lo tanto, en ambos casos es necesario desarrollar nuevas funcionalidades para lograr una herramienta más completa. De ahí que es importante evaluar la facilidad que presenta cada herramienta para ser extendida.

Primero, cabe destacar que una herramienta más difícil de extender requiere de mayor tiempo de desarrollo que una herramienta más fácil para lograr las mismas funcionalidades deseadas. En consecuencia, al seleccionar una herramienta más simple es posible dedicar el tiempo ahorrado a desarrollar nuevas funcionalidades o mejorar la calidad de la solución. De esta manera resultando en una herramienta más completa en la misma cantidad de tiempo al optar por la herramienta más fácil de extender.

Otra arista a considerar es la facilidad para entender el funcionamiento de la herramienta. Previo a implementar una funcionalidad, es necesario entender como funcionan los distintos componentes de la herramienta y cómo al modificarlos es posible lograr la funcionalidad buscada. En el caso que la herramienta presente complicaciones en este aspecto este proceso podría tardar más y, por lo tanto, atrasar la parte de desarrollo. De ahí que, una herramienta más fácil de entender y estudiar tiende a preferirse en gran medida sobre las que no. Cabe destacar que este proceso incluso en las herramientas más simples es bastante complejo y conlleva una cantidad considerable de tiempo de estudio. Pero que igual hay una diferencia importante en términos de dificultad entre herramientas, por lo que sigue siendo un factor de gran importancia.

En lo que respecta a GVM y Nerve, hay un claro ganador en términos de facilidad para extender. GVM presenta una gran cantidad de módulos, sistemas de comunicación y lenguajes de bajo nivel. Mientras que Nerve presenta una estructura simple más cohesiva, uniforme y con un lenguaje de programación de alto nivel más fácil de extender. Situándose así como una mucho mejor opción que GVM en este ámbito. Sin embargo, no basta con esta ventaja para descartar GVM como potencial escáner de vulnerabilidades debido a las otras ventajas que presenta la herramienta.

GVM por su lado incluye de forma nativa una mayor cantidad de los requisitos buscados en este trabajo que Nerve. Por lo tanto, al optar por esta herramienta no es necesario desarrollar tantas funcionalidades como su contraparte para lograr un resultado final satisfactorio. Sumado a esto, posee la gran ventaja de que su estructura puede modificarse para adecuarse a la estructura de la red de la universidad, lo que es altamente valorado por

la oficina. En conjunto todos estos aspectos no permite determinar un claro ganador entre las 2 principales alternativas mencionadas. Por lo que es necesario probar las herramientas para analizarlas en mayor profundidad y llegar a un veredicto final.

3.4.5. Pruebas GVM

Las pruebas apuntan a entender mejor el funcionamiento de las herramientas y sus características particulares. En el caso de GVM, estas pruebas apuntan principalmente a probar la arquitectura *Master-Sensor*, la cual se explica a continuación, y también a verificar la facilidad de uso de la herramienta.

El ambiente utilizado para las pruebas utiliza 2 máquinas virtuales con el sistema operativo Ubuntu 18.04 conectadas a una misma red local. Las máquinas no incluyen ningún *software* aparte de los incluidos en la instalación del sistema operativo y tratan de simular la estructura *Master-Sensor*. Donde una de las máquinas se comporta como el *Agente* o *Sensor* encargado de realizar las pruebas de vulnerabilidades. Y la otra como el *Master* encargado de configurar los escaneos y de desplegar la información resultante de las pruebas.

Dada la arquitectura que se pretende adoptar es necesario un método de instalación manual de los módulos y sus relaciones. El proceso involucra instalar algunos módulos en una de las máquinas y el resto de los módulos en la otra. Luego configurar y conectar todos los módulos entre sí. Como resultado se espera que una de las máquinas pueda controlar remotamente la otra para efectuar las pruebas de vulnerabilidades y que esta posteriormente envíe los resultados al nodo principal para desplegarlos al usuario.

La configuración antes mencionada es de interés para la OSI. Puesto que en el caso de funcionar permitiría separar la interfaz de control de la herramienta y los resultados del resto de los módulos. Luego estos podrían juntarse con otras interfaces de monitoreo para mantener todo lo relacionado a monitoreo de redes en un mismo lugar. Y así facilitar el manejo de la red en el caso que se disponga de múltiples herramientas diferentes.

La instalación parte por clonar el repositorio de cada módulo en la máquina adecuada, construir el módulo siguiendo los pasos de la documentación y luego conectarlos entre si mediante el protocolo de comunicación que corresponda. Sin embargo, en la práctica el proceso de instalación fue bastante más difícil de lo estipulado. Durante la instalación de los módulos se encontraron varios inconvenientes que impidieron instalar la herramienta correctamente bajo la configuración deseada.

Partiendo por que varios módulos entregaron errores al compilar el código. Estos errores generalmente estaban ligados a la versión de las librerías utilizadas, las cuales no habían sido actualizadas en los requerimientos de las versiones de github estables. En consecuencia, fue necesario actualizar gran parte del *software* y cambiar algunas configuraciones, lo que a su vez produjo nuevos problemas de compatibilidad entre las librerías y módulos. Finalmente, luego de varias iteraciones de cambios, se logró instalar todos los paquetes y sus dependencias correctamente en cada una de las máquinas. No obstante, no se lograron conectar correctamente los módulos con todas sus interacciones debido a nuevos errores en los protocolos de comunicación. Dando como resultado una herramienta poco funcional e insuficiente para el contexto en el que debe desempeñarse. Finalmente se decidió descartar

la alternativa en vez de seguir gastando tiempo en lograr que funcione.

3.4.6. Pruebas Nerve

Las pruebas realizadas en la herramienta Nerve se enfocan en validar 3 principales propiedades, listadas a continuación:

1. Que sea fácil de instalar.
2. Que sea fácil de navegar.
3. Que se puedan encontrar vulnerabilidades correctamente.

Similar a GVM se utilizó una máquina virtual con un ambiente Ubuntu 18.04 conectada a una red local para instalar el *software*. Además en la misma máquina se configuró un servicio FTP no seguro, con falta de credenciales, que puede ser explotado. Y se espera que la herramienta sea capaz de detectar la vulnerabilidad en el servicio local.

A diferencia de GVM, la instalación de Nerve es considerablemente más fácil, en gran parte debido a que no es necesario separar los diferentes módulos de la herramienta en máquinas distintas. Para instalar basta con clonar el repositorio y correr un archivo de instalación presente en el proyecto. El archivo luego se encarga de instalar todas las librerías, dependencias, correr procesos y dar una configuración básica a la herramienta, dando por finalizada la etapa de instalación.

Como paso siguiente, se intentó ejecutar un prueba de vulnerabilidad en la red local. Esta se llevo a cabo utilizando el perfil de *admin* creado en la instalación y navegando a la interfaz de *escaneo rápido* de la herramienta, dando inicio al proceso. Con esto bastó para detectar la falla en el sistema FTP y un par de otras configuraciones sospechosas de la máquina. Resultando en una prueba exitosa para la herramienta.

Posterior a los resultados del escaneo se exploraron las distintas interfaces de la aplicación web con sus variadas funcionalidades. En general, todas las interfaces son muy intuitivas y fáciles de navegar, con un diseño limpio y minimalista. Sumado a esto, la herramienta también cumple en su mayoría con las funcionalidades que dice tener. Hay una única excepción a esto, la herramienta no presenta una opción para agendar los escaneos a pesar de incluir la opción *Schedule Scan* en su configuración. Esta última solo sirve para efectuar escaneos inmediatos o continuos. Si bien esta falta le genera un problema a la oficina, sigue siendo viable desarrollar la funcionalidad durante la etapa de desarrollo y así lograr lo buscado.

3.4.7. Herramienta escogida

Los resultados obtenidos de las pruebas realizadas favorecen en gran medida a Nerve sobre GVM. La segunda herramienta presenta inconvenientes relacionados al proceso de instalación y configuración, para los cuales es necesario dedicar una gran cantidad de tiempo para solucionar. Tiempo que podría aprovecharse en mejorar la primera herramienta en el caso que se seleccionase esta. Además, es posible que los problemas sigan persistiendo a futuro e impacten negativamente a la oficina. Por lo que GVM deja harto de desear en

cuanto a la compilación e instalación manual de los módulos. En consecuencia, a pesar de que la herramienta presenta claras ventajas en términos de flexibilidad y cantidad de funcionalidades, estas no son suficientes como para contrarrestar las problemas asociadas a su instalación y extensión.

En contraste, Nerve es una de las herramienta que mejor se desempeña en los atributos previamente mencionados. Estas capacidades son muy valoradas para efectos de este trabajo, y son suficientes para justificar la elección de la herramienta aún cuando presenta ciertos problemas en términos de desempeño, mantención y menor cantidad de funcionalidades. Esto último se contraresta en gran medida debido a la facilidad para extender la herramienta. Lo que permite desarrollar las funcionalidades faltantes en una cantidad razonable de tiempo y así lograr una herramienta final satisfactoria para la oficina.

En conclusión, Nerve a pesar de no ser la herramienta más completa y poseer ciertos problemas en diferentes ámbitos, se posiciona como la mejor herramienta para este trabajo. Gran parte de esto se debe a la facilidad de uso de la herramienta, facilidad de extender y facilidad de instalación. Estas características permiten ahorrar una cantidad considerable de tiempo, que luego puede utilizarse para mejorar la solución final y así cumplir con los requisitos de la oficina en el tiempo acotado del trabajo.

3.5. Selección formato *scripts*

Por lo general, al seleccionar la herramienta encargada de realizar las pruebas de vulnerabilidades también se selecciona indirectamente el formato de los *scripts* a ejecutar. En la mayoría de los casos las herramientas solo soportan un único formato nativo de *script* y solo queda adaptarse a este. En cuanto a la herramienta Nerve, esta también soporta solo un único tipo de *script*. Pero a diferencia del resto, posee la ventaja de que es *open source*, lo que abre las puertas a introducir un nuevo formato si se opta por trabajar en eso. En consecuencia, dependiendo de la dificultad y beneficios puede ser una opción viable migrar a un nuevo formato de ejecución de *scripts*.

Nerve utiliza por defecto un formato de *scripting* propio basado en Python. El formato se encuentra bien estructurado, es relativamente simple de entender y posee las ventajas del lenguaje Python. Sin embargo, presenta una gran debilidad en comparación con otras alternativas; no es un formato muy popular. Por lo tanto, gran parte de los *scripts* que han sido desarrollados en el formato ya se encuentran incorporados en la herramienta. De ahí que, no genera tanto valor desarrollar una funcionalidad que permite agregar *scripts* en este formato si la mayoría de los *scripts* ya se encuentran incorporados. En contraste, al seleccionar un formato más popular podrían incorporarse muchos más *scripts* de diferentes tipos, aprovechando mucho mejor la capacidad para incorporar *scripts* a la herramienta.

Dada esta situación se explora la idea de extender la herramienta para incorporar nuevos formatos de *scripts*. La oficina al inicio del trabajo expresó interés en el formato *Nmap Scripting Engine*(NSE), en el cual ya se encontraban previamente familiarizados [5]. A diferencia del formato presente nativamente en la herramienta, NSE corresponde a un formato comúnmente utilizado en el área de ciberseguridad y generalmente considerado como uno de los principales formatos de *scripting*. En consecuencia, existen varios *scripts*

escritos en el formato NSE en la herramienta Nmap e internet, los cuales podrían agregarse a Nerve si es requerido. Esto hace que el formato NSE sea preferible al formato por defecto de la herramienta.

El formato *Nmap Scripting Engine*(NSE) como lo indica su nombre fue desarrollado para ser usado en conjunto con la herramienta Nmap [7]. Nmap es necesario para ejecutar los *scripts* en el formato pero trae consigo otros beneficios como escaneo de puertos, mapeo de la red y determinar dispositivos disponibles en una red. El lenguaje en el cual se basa, Lua, soporta múltiples técnicas de programación y se caracteriza por ser simple de trabajar y extender [6]. Por su lado NSE incluye múltiples librerías que permiten realizar labores como detección de redes, detección sofisticada de versiones, explotación de vulnerabilidades, entre otros. Utilizando estas librerías es posible desarrollar todo tipo de *scripts* adaptándose a las distintas vulnerabilidades que afectan la seguridad de la red.

El formato destaca además por su facilidad para ser incorporado en el trabajo. Puesto que algunos de los componentes de Nmap ya son utilizados por la herramienta. En particular, cumple un rol fundamental para realizar actividades de detección de servicios en puertos y mapeo de la red objetivo. Sumado a esto, el *output* de la ejecución de *scripts* utiliza el formato XML, el cual permite trabajar fácilmente con la información. Todos estos beneficios y características posicionan al formato NSE como la mejor alternativa en cuanto a formato de *scripts*. Por lo que se opta por introducir el formato NSE a la herramienta.

Capítulo 4

Desarrollo de la solución

En el presente capítulo se discute la implementación de las principales requisitos del trabajo. De forma complementaria, se explica la arquitectura del *software*, librerías utilizadas, problemas encontrados y decisiones de diseño tomadas. El repositorio con todos los cambios se encuentra en la plataforma *github* y se puede acceder a través del enlace: https://github.com/TomasTorresB/nerve.

4.1. Requisitos a abordar

A fines del capítulo anterior se selecciona el escáner de vulnerabilidades que mejor se acomoda a las condiciones del trabajo. Antes de pasar directamente a implementar las nuevas funcionalidades, es necesario determinar cuales de los requisitos definidos inicialmente faltan por abordar. Puesto que algunas de las funcionalidades definidas en un comienzo ya se encuentran implementadas en la herramienta seleccionada. Un análisis total de los requisitos del trabajo se presenta en la tabla A.1 del anexo. Los marcados con "—" o "X" no vienen implementados por defecto en la herramienta y son abordados a lo largo de esta sección.

Dada la gran cantidad de requisitos no es viable profundizar en cada uno de ellos, en cambio, se opta por analizar las categorías a las cuales pertenecen. En la presente sección, se explica a grandes rasgos las soluciones adoptadas y trabajo necesario para alcanzar los requisitos definidos dentro de cada categoría. En las siguientes secciones del capítulo se profundiza sobre la implementación de cada uno de estos métodos y las técnicas utilizadas.

Partiendo por los requisitos pertenecientes a la categoría *Automatización*, es preciso destacar que la herramienta seleccionada pone gran importancia en este ámbito. Sin embargo, las funcionalidades de este tipo fueron diseñadas para los *scripts* nativos de la herramienta y no otros tipos. En consecuencia, para satisfacer los requisitos pertenecientes a esta categoría es necesario extender las funcionalidades preexistentes para soportar el formato NSE. En otras palabras, basta con integrar el formato NSE en la herramienta y reutilizar las funcionalidades ya disponibles para automatizar el proceso de ejecución de *scripts*. Este problema se aborda posteriormente en la sección *Integración scripts NSE*.

Respecto a los requisitos pertenecientes a la categoría *Incorporación scripts*, dependiendo del tipo hay 2 distinciones claves que realizar. Por un lado, un grupo de requisitos se centra

en la transformación de *scripts* al formato NSE. Estos requisitos no se ven afectados por el tipo de herramienta seleccionada, ya que la transformación de *scripts* es un proceso aparte. En consecuencia, todos los requisitos de este tipo se mantienen activos incluso después de seleccionar el escáner de vulnerabilidades. Las especificaciones del proceso de transformación se detallan en la sección *Transformación scripts*.

En el otro extremo se encuentran los requisitos relacionados a la incorporación de *scripts* en la herramienta, los cuales también deben ser desarrollados casi en su totalidad. Gran parte de esto se debe a que la herramienta no provee facilidades para incorporar nuevos *scripts* a pesar de ser una herramienta *open source*. De ahí que es necesario incorporar las funcionalidades asociadas a los requisitos vigentes. El desarrollo relacionado a estos requisitos se incluye en la sección *Integración scripts NSE*.

Luego, en cuanto a los requisitos pertenecientes a la categoría *Facilidad de uso*, cabe recordar que una de las grandes ventajas de Nerve sobre las otras herramientas es la facilidad para manejarla. En consecuencia, por defecto la herramienta aborda casi todos los requisitos pertenecientes a la categoría, a excepción del requerimiento que las interfaces se encuentren en español. Por este motivo, parte del trabajo también se centra en este último aspecto y se detalla en la sección *Traducción interfaces*.

Con respecto a los requisitos complementarios, la incorporación de estos en la herramienta varía mucho dependiendo del requisito. Algunos se encuentran incorporados nativamente en la herramienta, otros parcialmente y otros no. El trabajo necesario para lograr los requisitos faltantes, por lo general, puede incluirse en algunas de las secciones previamente mencionadas, a excepción del requisito *Agendado de escaneos*, el cual dispone de su propia sección.

Los últimos requisitos pertenecientes a la categoría *Entrega de la solución* se abordan posterior al desarrollo de la solución. Por consiguiente, en vez de incluirse en esta sección se discuten más adelante, particularmente en el capítulo *Incorporación de la solución*.

Antes de pasar directamente a las explicaciones del trabajo es necesario entender superficialmente la estructura de la herramienta y su funcionamiento interno. Este conocimiento es relevante para entender las funcionalidades desarrolladas y las decisiones tomadas.

4.2. Arquitectura Nerve

Nerve se divide en varios módulos que le permiten a la herramienta desplegar la interfaz gráfica al usuario, procesar las solicitudes, realizar las pruebas de vulnerabilidades, guardar información, entre otros.

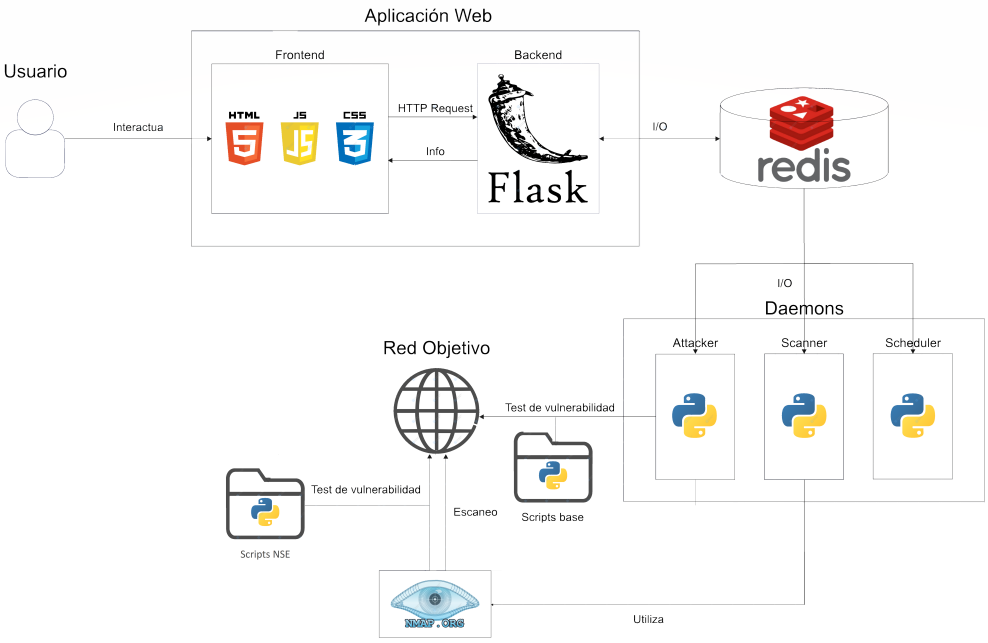

Figura 4.1: Arquitectura herramienta Nerve

En las siguientes secciones del capítulo se explican los distintos módulos y componentes de la arquitectura de la Figura 4.1.

4.2.1. Aplicación web

Se divide en *frontend* y *backend*. El *frontend* está escrito en los lenguajes HTML, CSS y Javascript. Y corresponde principalmente a las interfaces de la aplicación, también referenciadas como *templates* más adelante. Aquí se registran todas las interacciones de los usuarios, las que posteriormente se envían al *backend* para ser procesadas.

El *backend* se encarga de manejar las solicitudes del usuario. Por lo que su principal enfoque es retornar la información solicitada al *frontend* para poder desplegarla. En el caso que no se disponga de la información se corren los procesos y algoritmos necesarios para obtenerla o se consulta por ella con el resto de los módulos. Para el manejo de las solicitudes y la entrega de la información a los *templates* se utiliza el *framework Flask* [24]. El cual permite fácilmente entregar la información correcta a los *templates*.

4.2.2. *Daemons*

Para la ejecución de las pruebas de vulnerabilidades el *backend* de la aplicación se apoya en unos procesos llamados *daemons*. En computación, un *daemon* es un proceso que corre por atrás sin parar, en el cual el usuario generalmente no tiene control. La herramienta posee 3 principales *daemons* que están continuamente corriendo, correspondientes a:

- *Scheduler*: Una vez que se registra un nuevo escaneo se encarga de filtrar la información relevante de acuerdo a las configuraciones escogidas y de guardar la información en la

base de datos.

- *Scanner*: A partir de las configuraciones de *Scheduler* realiza un análisis de los puertos vulnerables de los servicios que se quieren atacar. Utiliza la herramienta Nmap como base para el análisis de puertos.
- *Attacker*: Supervisa la ejecución de las pruebas de vulnerabilidades.

4.2.3. Redis

Redis es la principal base de datos utilizada en el proyecto [25]. Es un *software open source* del tipo *in-memory*, esto quiere decir que depende primariamente de la memoria del sistema para el almacenaje de datos, a diferencia de otras bases de datos que se basan en discos o dispositivos SSD. A raíz de esto, es que es una de las bases de datos más rápida y con mejor desempeño.

El *software* se caracteriza por almacenar estructuras de datos como *hashes*, listas, *sets*, etc. Además es fácil de extender y de programar, por lo que es una alternativa conveniente para proyectos en donde no es necesario realizar *queries* intrincadas o cuando se valora en gran medida la velocidad.

Dentro de Nerve, Redis es usado como un cache para almacenar información de los escaneos. La gran facilidad para acceder a esta base de datos le permite al resto de los módulos trabajar con la información fácilmente y posibilitar el traspaso de información entre los módulos. La información almacenada no es mucha por lo que es posible sacar provecho de la memoria del sistema para obtener un mejor desempeño.

4.2.4. Nmap

Nmap es una herramienta comúnmente utilizada en el área de la ciberseguridad, se caracteriza por su capacidad de analizar puertos vulnerables en sistemas y por incluir *scripts* que permiten efectuar pruebas de vulnerabilidades en una gran cantidad de servicios diferentes [7].

Existen algunas librerías en Python que permiten ocupar las múltiples funcionalidades de Nmap. La herramienta Nerve base incluye la librería *python-nmap* para hacer analizar los puertos del objetivo de un ataque [26]. Este trabajo además se aprovecha de otras funcionalidades de la librería para ejecutar los *scripts* NSE desarrollados por la OSI y cualquier otro del tipo NSE. En suma, Nmap es utilizado por los *daemons attacker* y *scanner* para realizar ataques y escaneos de red respectivamente.

4.2.5. Flujo ejecución escaneos

La arquitectura mencionada está desarrollada principalmente con la intención de realizar escaneos de vulnerabilidades. En la presente sección se explica el flujo entre los diferentes módulos de la arquitectura para completar la ejecución de las pruebas. El proceso descrito a continuación utiliza como referencia la Figura 4.1 de la arquitectura.

El proceso inicia cuando el usuario interactúa con la interfaz gráfica de la herramienta. Esto le permite comenzar la ejecución de escaneos y visualizar información de interés. Las interacciones luego son enviadas al *backend* de la herramienta para ser procesadas. El *backend* por su parte interpreta la información recibida y determina los próximos pasos a seguir. Si se quieren visualizar ciertos datos se los entrega al *frontend* para que el usuario los pueda visualizar. En el caso que se quiera agendar un escaneo, guarda la solicitud con su respectiva configuración en la base de datos para que el resto de los módulos pueda trabajar en la tarea.

Una vez que la configuración del escaneo llega a ser guardada en Redis, inmediatamente los deamons son alertados y comienzan sus funciones. Inicia primero el *daemon Scheduler* agendando las direcciones IPs próximas a ser atacadas y cambiando el estado de la sesión para indicar el indicar el inicio de un escaneo.

En segundo lugar el *daemon Scanner* analiza las direcciones agendadas con el apoyo de Nmap. De este modo, se determinan los puertos vulnerables y los tipos de servicios que corren en los objetivos. En el caso que se detecte alguna anomalía se guarda la información en la base de datos.

Luego de recopilar toda la información del objetivo empieza el ataque al servicio por parte del *daemon Attacker*. Este utiliza la información encontrada por *Scanner* en conjunto con la configuración entregada por el usuario para personalizar los ataques a las diferentes redes. Los ataques se efectúan mediante la ejecución de diferentes tipos de *scripts* de la herramienta con parámetros personalizados. En el caso que los *scripts* sean del tipo NSE se utiliza la herramienta Nmap para lograr su ejecución. Al momento de encontrar una vulnerabilidad en alguno de los sistemas se guarda la información en la base de datos.

Finalmente, el *backend* de la aplicación saca toda la información asociada a las vulnerabilidades encontradas y se la entrega al *frontend* para que muestre los resultados al usuario. De esta forma se completa el proceso de ejecución de las pruebas de vulnerabilidades.

En las próximas secciones del capítulo se detalla el trabajo desarrollado para abordar los requisitos establecidos. Varias de las explicaciones se apoyan en los componentes y flujos mencionados durante esta sección.

4.3. Integración *scripts* NSE

La integración de los *scripts* en la herramienta apunta a resolver tanto los requisitos pertenecientes a la categoría *Automatización* y parte de los de *Incorporación de scripts*. Primero, recordar que la herramienta seleccionada ya incluye las funcionalidades necesarias para automatizar el proceso de ejecución de *scripts*. Sin embargo, esta solo se centra en los *scripts* pertenecientes al formato nativo de la herramienta. En consecuencia, se busca adaptar el proceso de ejecución de *scripts* para también soportar el formato NSE utilizado en este trabajo. Por ende, la integración correcta del formato NSE se encuentra directamente relacionada a la capacidad de automatizar el proceso de ejecución de *scripts*.

Por otro lado, la integración de *scripts* NSE, como su nombre lo indica, también abarca la incorporación de nuevos *scripts* a la herramienta. Esta funcionalidad es crucial para utilizar los *scripts* OSI y cualquier otro del tipo NSE en las pruebas de vulnerabilidades. El detalle asociado se especifica al final de la sección.

4.3.1. Automatización proceso ejecución de *scripts*

El proceso para ejecutar los *scripts* de la herramienta involucra interactuar con diferentes módulos y componentes que se mencionan en la sub-sección *Flujo ejecución escaneos*. Durante este proceso hay 4 interacciones de gran importancia para la inclusión de *scripts* NSE. Estas corresponden a:

1. Determinar cuando ejecutar un *script*.
2. La ejecución del *script*.
3. Determinar si existe una vulnerabilidad.
4. Guardar la información resultante de la prueba.

A continuación, se discuten las modificaciones a realizar a cada uno de los elementos de la lista para poder incorporar los *scripts* NSE en la herramienta.

4.3.1.1. Determinar que *scripts* ejecutar

Al momento de iniciar un escaneo es importante tener claro cuáles *scripts* van a ser ejecutados durante las pruebas de vulnerabilidades. En otras palabras, cuáles *scripts* se utilizarán durante las pruebas. Para esto el *daemon Scheduler* de la herramienta se apoya en el *Nivel de agresividad* de un escaneo. Este valor numérico, dado por el usuario en las configuraciones de un escaneo, indica el potencial impacto de la prueba sobre el objetivo. Un escaneo con cierto nivel de agresividad implica la ejecución de *scripts* con niveles de agresividad iguales o menores al definido. Los *scripts* con altos niveles de agresividad tienden a afectar el desempeño o generar la caída de los servicios, en consecuencia, es necesario establecer un nivel de agresividad adecuado dependiendo del objetivo. Y a la larga, es este valor el que determina los grupos de *scripts* que deben ejecutarse durante una prueba.

Internamente cada *script* base de la herramienta presenta un campo con su propio nivel de agresividad. Luego cuando un usuario efectúa un escaneo de cierto nivel, se chequean los valores internos de todos los *scripts* y se seleccionan los que cumplen con el criterio establecido. De este modo, se filtran los *scripts* que deben ser ejecutados del total.

El problema al integrar los *scripts* NSE, es que estos no incluyen un campo similar al *Nivel de Agresividad*. Como no se proporciona esta información no hay forma de determinar cuando ejecutar un *script* siguiendo las reglas de la herramienta. Similarmente, hay otros campos presentes en los *scripts* base tampoco se incluyen en los *scripts* NSE y que también generan otros problemas, provenientes de la falta de información.

Primero, se busca que el campo *Nivel de agresividad* no sea indispensable para determinar que *scripts* ejecutar. Para lograr esto se le asigna el máximo nivel de agresividad

a todos los *scripts* sin el campo. De este modo, los *scripts* solo serán ejecutados cuando el usuario ingrese la configuración de ejecución más agresiva. Puesto que bajo dicha configuración es aceptado ejecutar cualquier tipo de *script* independiente del impacto que pueda tener. Por lo que no hay problema al incluir *scripts* con niveles de severidad muy altos o indeterminados.

Pese a que este enfoque permite ejecutar los nuevos *scripts*, al utilizar la configuración más agresiva de ataque, no es suficiente para abordar todas las aristas del problema. Utilizando la configuración previamente mencionada todos los *scripts* incorporados solo pueden ser ejecutados utilizando el mayor nivel de agresividad. Sin embargo, para la oficina también es de interés ejecutar *scripts* NSE en configuraciones menos agresivas. Por lo tanto, para satisfacer esta necesidad se implementa un segundo método más personalizado.

El método requiere que el usuario indique el nivel de agresividad del *script* a incorporar. En otras palabras se utilizan los mismos campos que poseen los *scripts* nativos de la herramienta en los *scripts* NSE. Por lo tanto, este método se basa en que el usuario o desarrollar del *script* incluya la información necesaria en los campos del *script*. Posteriormente, la herramienta lee la información de estos campos y reutiliza el mismo algoritmo de filtrado, basado en niveles de agresividad, para filtrar los *scripts* pertenecientes al formato NSE. Sin embargo, esto se basa en la idea que los usuarios y creadores de los *scripts* rellenan la información de los campos en cuestión, lo que muchas veces no ocurre por diferentes motivos. Por lo que el primer método, en donde se le asigna el máximo nivel a los *scripts*, sigue siendo relevante en el resto de los casos.

4.3.1.2. Campos relevantes *scripts* nativos

Similarmente, los *scripts* nativos de la herramienta poseen campos adicionales a los del formato NSE que también debe ser abordados. El otro campo que destaca es el *Nivel de severidad*, de gran valor para la oficina. Este indica que tan grave es la vulnerabilidad encontrada por un *script* determinado.

Para efectos de la ejecución de los *scripts* su impacto es mínimo. Debido a que afecta únicamente los resultados desplegados en las componentes visuales En consecuencia, para mitigar los efectos de cuando no se proporciona este campo, basta con indicar en las interfaces que no se dispone de esta información. Para la herramienta esto es equivalente a clasificar los *scripts* con el nivel de severidad *Indeterminado* y editar las interfaces para que se pueda desplegar la información. De este modo, es posible incorporar nuevos *scripts* NSE a la herramienta sin la necesidad del campo nivel de severidad como se muestra en la Figura 4.2

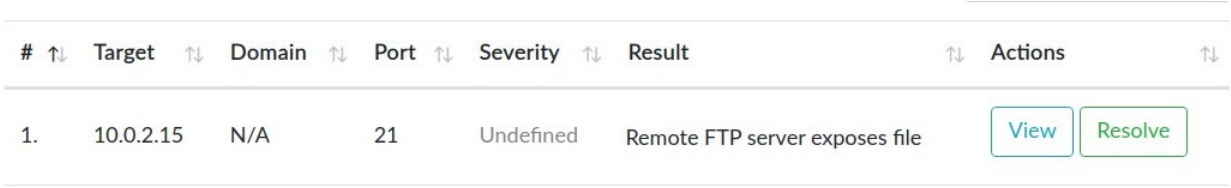

Figura 4.2: Nivel de Severidad Indeterminado

Por último, para abordar el resto de los campos faltantes no es necesario realizar grandes

cambios a la herramienta. Por lo general, la mayoría de los campos solo brindan información complementaria a la ejecución del *script*, la cual posteriormente es mostrada al usuario. Debido a que estos campos no interfieren con el resto de las funcionalidades es posible omitirlos sin grandes repercusiones. En tal caso, simplemente no se despliega la información al usuario o se indica la falta de esta información, similar al nivel de severidad. Eliminando así la dependencia de los campos en la ejecución de *scripts*.

En suma, para utilizar todas las funcionalidades disponibles en la herramienta es necesario introducir nuevos campos a los *scripts* NSE, basados en los campos de los *scripts* nativos. La información contenido en estos campos permite utilizar la estructura ya implementada y cumplir con todas las funcionalidades. Sumado a esto, se incluyen opciones para añadir *scripts* sin los nuevos campos, pero con ciertas limitaciones. De esta manera, es posible introducir cualquier *script* del tipo NSE asumiendo algunas restricciones y, a la vez, mantener todas las funcionalidades de la herramienta vigentes.

4.3.1.3. Ejecución *scripts*

Luego de solucionar los problemas originados por la falta de campos, el siguiente paso a abordar en el proceso de ejecución de *scripts* corresponde a ejecutar los *scripts* directamente. Esto es efectuado por el *daemon Attacker* y en el caso de los *scripts* NSE es posible gracias a la librería *Python-nmap* [26]. Esta librería permite al *backend* de la aplicación interactuar con Nmap y así organizar la ejecución de los *scripts* NSE. De este modo, es posible iniciar la ejecución de los *scripts* con los parámetros ingresados por el usuario desde la herramienta.

Al momento de realizar la ejecución también es necesario considerar unas propiedades de los *scripts* NSE llamadas *reglas*. Las reglas son condiciones que afectan la ejecución de un *script*. Por ejemplo, una regla puede indicar que un *script* solo puede ser ejecutado en un puerto determinado. En tal caso, al atacar otro puerto diferente se termina la ejecución inmediatamente. Estas condiciones pueden ocasionar inconsistencias entre las configuraciones entregadas por el usuario y las reglas propias del *script*. De ahí que es necesario priorizar una de ellas por sobre la otra. En este caso se opta por priorizar las reglas sobre las configuraciones de los usuarios, puesto que las reglas son desarrolladas por los creadores de los *scripts* y generalmente abordan diferentes casos específicos. En algunos casos ignorar las reglas puede resultar en problemas durante la ejecución.

4.3.1.4. Determinar existencia de vulnerabilidades

El próximo paso en el proceso corresponde a determinar si luego de una prueba se encontró una vulnerabilidad. Esta información es crítica para el usuario ya que es de suma importancia determinar si el sistema atacado es vulnerable. Si bien esto puede sonar trivial, un gran problema de los *scripts* NSE es que no incluyen un formato de salida estándar de sus resultados. Muchas veces se ocupan librerías que definen de cierta forma el formato de salida de cada *script*, pero al final, cada desarrollador define el formato a utilizar. Esto hace imposible crear un método universal que permita determinar cuando el resultado corresponde a una vulnerabilidad. De ahí que, se busca implementar un método, con ciertas limitaciones, para determinar cuando el resultado de un *script* corresponde a una vulnerabilidad.

El problema se abordó de 2 diferentes maneras. Similar a los problemas asociados a los campos de los *scripts*, uno de los métodos involucra trabajo adicional por parte del usuario pero permite utilizar todas las funcionalidades de la herramienta. Mientras que el otro método no requiere de intervenciones adicionales y permite incluir cualquier tipo de *script* pero con funcionalidades limitadas. A continuación se explican ambos métodos.

El primer método se encarga de entregar soporte a un grupo selecto de *scripts* dentro de la herramienta. Esto se logra analizando los posibles resultados de un *script* e identificando cuales de los *outputs* corresponden a una vulnerabilidad. En otras palabras, modificar la herramienta para que esta pueda detectar los diferentes *outputs* de un *script*. En tal caso los *scripts* pasan a comportarse similar a los de Python, en el sentido que la herramienta es capaz de determinar cuando se encuentra una vulnerabilidad y la almacena adecuadamente. La gran desventaja es que estos requieren de intervención manual para procesar los diferentes resultados. Pero de este modo es posible saber con seguridad cuando se encuentra una vulnerabilidad.

La otra alternativa consiste en agregar una nueva categoría *Potencial*, para indicar que no se ha podido verificar si el resultado de un *script* corresponde a una vulnerabilidad. Al utilizar este método, una vez finalizado el escaneo, se despliegan todos los resultados de las pruebas al usuario pero señalando las vulnerabilidades que pertenecen a este categoría. Por ende, queda en manos del usuario discernir cuales representan una amenaza y cuales no. La Figura 4.3 y la Figura 4.4 corresponden algunas de las interfaces afectadas por este cambio.

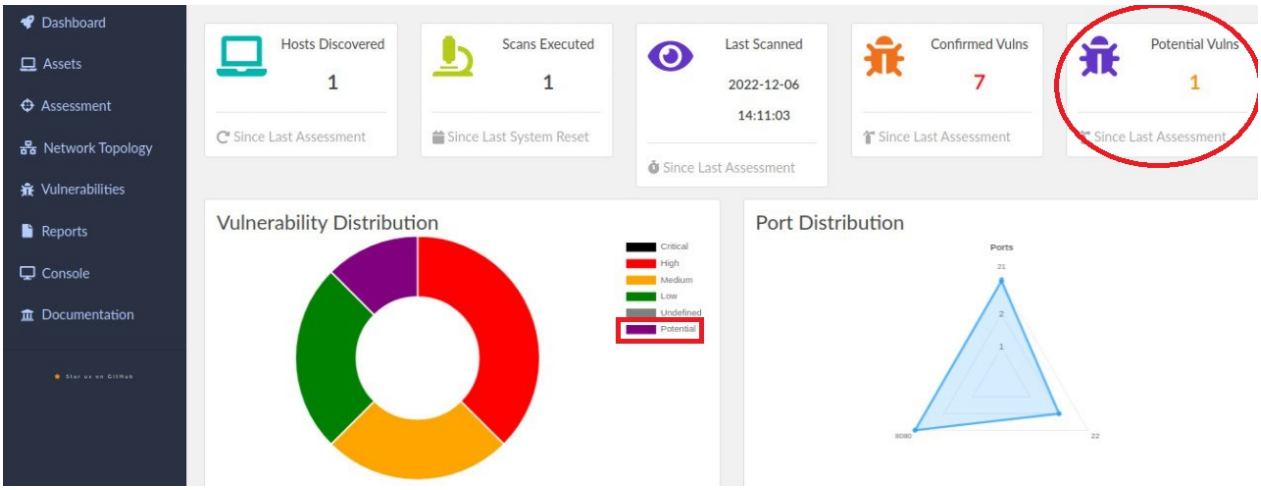

Figura 4.3: *Dashboard* con Vulnerabilidad Potencial

#	Objetivo	Dominio	Puerto	Severidad	Resultados	Acciones
1.	10.0.2.15	N/A	21	Potencial	Accounts ftpuserftpuser Valid credentials Statistics Performed 40 guesses in 7 seconds average tps 57	Ver Resolver

Figura 4.4: Detalle Vulnerabilidad Potencial

La gran ventaja de este método es que abre la posibilidad de trabajar con cualquier tipo de *script*, no solo los que se encuentran con soporte en la herramienta. A cambio, puede que

se muestre información que no es relevante para el usuario. Lo que puede ser inconveniente al trabajar con una gran cantidad de resultados de este tipo. De todos modos, si el usuario quiere evitar esta inconveniencia puede preferir el otro método más personalizado.

4.3.1.5. Guardar vulnerabilidades encontradas

Por último, luego de encontrar una vulnerabilidad es necesario guardar la información relevante en la base de datos para su uso posterior en los otros módulos de la herramienta. En el caso de los *scripts* NSE es posible utilizar el mismo formato que el resto de los *scripts* para almacenar las vulnerabilidades, sin la necesidad de realizar mayores cambios a la estructura interna. De esta manera, el resto de los módulos puede trabajar con las vulnerabilidades de los nuevos *scripts* sin mayores problemas.

Todos estos cambios permiten a la herramienta automatizar el proceso de ejecución de *scripts* NSE. Y de esta manera se logra abordar uno de los principales requerimientos del trabajo. En el resto de la sección se aborda el segundo requisito relacionado a la integración de *scripts*, correspondiente a la incorporación de nuevos *scripts* a la herramienta.

4.3.2. Agregar nuevos *scripts*

El trabajo asociado a la incorporación de los *scripts* en la herramienta puede dividirse en 2 principales tareas. En primer lugar, es necesario implementar un método que permita a los usuarios agregar nuevos *scripts*. Sumado a esto, se busca introducir un mecanismo que permita a los usuarios utilizar argumentos personalizados al momento de ejecutar los *scripts*. Así los operadores de la herramienta pueden introducir nuevos *scripts* y ejecutarlos a su gusto.

El método para incorporar nuevos *scripts* inicialmente considera implementar una nueva interfaz con opciones para subir los archivos de los *scripts*. Sin embargo, el trabajo necesario para desarrollar tal funcionalidad no se justifica, dada la existencia de otras alternativas más simples pero igual de efectivas. En particular, basta con que el usuario incorpore directamente los *scripts* en una carpeta determinada de la herramienta, desde la cual el *software* posteriormente pueda leer los archivos. En contraste con la propuesta inicial esta requiere de bastante menos desarrollo para implementar, pero posee la desventaja de que el usuario debe interactuar con componentes fuera de la interfaz gráfica. Sin embargo, el gran ahorro de tiempo asociado a esta alternativa hace que esta sea la opción preferible para este trabajo. En consecuencia, se opta por esta última con la condición de incluir un conjunto de instrucciones para subir los archivos, con el propósito de guiar al usuario y facilitar el proceso.

Introducir nuevos *scripts* tiene otras repercusiones en el funcionamiento de la herramienta. La repercusión más notoria tiene relación con los argumentos de los nuevos *scripts*. Puesto que la herramienta no es capaz de determinar cuales argumentos acepta cada *script* NSE en particular. Esto complejiza en gran medida el manejo de esta información. Puesto que no es posible indicarle al usuario cuales argumentos utiliza cada *script*, y en consecuencia, este no es capaz de proveer los valores de los argumentos. Dificultando el uso de argumentos personalizados durante la ejecución de las pruebas.

Para lidiar con este problema se opta por un sistema de responsabilidades compartidas

entre la herramienta y el usuario. En particular, se deja en manos del usuario determinar que parámetros utiliza cada *script*. Lo que no es difícil desde el punto de vista humano al leer el código o descripción de un *script*. Luego, basta ingresar los valores de los argumentos en el archivo de configuraciones de la herramienta. Así, la herramienta simplemente utiliza esta información durante la ejecución, independiente si es válida o no. De esta manera, al modificar el archivo de configuraciones es posible ejecutar los *scripts* con argumentos personalizados.

La implementación de la funcionalidad se basa en la correcta nomenclatura de los argumentos en las configuraciones. Se utiliza un algoritmo que compara los nombres de los argumentos del archivo de configuraciones con el nombre del *script* para determinar los parámetros relevantes. En consecuencia, es vital nombrar correctamente los argumentos para que la herramienta los reconozca correctamente.

Esta opción permite centralizar todos lo relacionado con parámetros en un mismo lugar. Además, permite configurar cualquier parámetro de cualquier manera, lo que brinda flexibilidad en la ejecución de los *scripts* y permite adecuarse a cualquier tipo de situación que pueda surgir. Sin embargo, posee la desventaja de que es necesario interactuar con archivos fuera de las interfaces de la herramienta. Pero dado el gran ahorro de tiempo asociado a esta alternativa es que se justifica la solución adoptada. Para ayudar al usuario se incluyen notas del formato necesario para nombrar los parámetros de los *scripts* y las instrucciones, paso a paso, de como editar el archivo de configuración. Considerando todos estos aspectos, la alternativa escogida mantiene un buen balance entre simpleza, facilidad de implementar y cumplimiento de sus objetivos.

4.4. Transformación *scripts*

En la presente sección se detalla el proceso asociado a transformar un *script* OSI al formato NSE. En particular para este trabajo se selecciona un *script* encargado de extraer información de servicios FTP poco seguros. El proceso de transformación del *script* corresponde principalmente a replicar los mismos pasos del *script* original en el formato NSE. El proceso además involucra extenso testeo, inclusión de información complementaria e integración del *script* en la herramienta.

El objetivo del *script* es extraer información de un archivo disponible en un servidor FTP no seguro. El archivo en cuestión se llama *user.cfg* y la información buscada corresponde a las líneas del archivo que contengan los caracteres *tftp*, *password* o *write*. El *script* recibe 3 parámetros opcionales del usuario, los dos primeros corresponden a las credenciales del logeo al servicio FTP y el tercero corresponde al directorio en el cual se encuentra el archivo *user.cfg*. En el caso que no se entreguen los parámetros, se prueba la conexión al servicio sin credenciales y se busca el archivo en el directorio principal. A continuación se detallan los pasos que sigue el *script* para lograr su objetivo:

1. Establecer conexión con el servidor FTP usando credenciales.
2. Navegar al directorio relevante.
3. Buscar el archivo *user.cfg*.

4. Leer todas las líneas del archivo que coincidan con las palabras claves.

5. Retornar todas las líneas con coincidencias al usuario.

El *script* se apoya fuertemente en el uso de *sockets*. NSE incluye varios tipos de *sockets* fáciles de trabajar. Para este *script* en particular es relevante un *socket* tipo FTP, el cual permite comunicarse con el servidor. De esta manera, es posible efectuar diferentes comandos en el servidor FTP enviando la información a través del *socket*. El *script* principalmente se encarga de enviar comandos al servidor simulando los pasos anteriormente descritos y de procesar las respuestas. De esta manera, es posible realizar peticiones al servidor hasta encontrar algún error u obtener algún resultado.

El *script* desarrollado pone especial cuidado en el procesamiento de la información del servidor, en vista que, el servidor FTP puede responder de múltiples maneras dependiendo de las configuraciones del servidor, la estructura de los archivos y la versión del servicio. Por lo tanto, después de cada petición se verifican múltiples casos bordes en función de las posibles respuestas. Para identificar todos los posibles casos se realizaron varias pruebas y se utilizaron como referencia otros *scripts* FTP de la herramienta Nmap que abordan casos similares. De este modo, se espera que el código resultante sea robusto frente a diferentes escenarios.

Una vez desarrollado el *script*, se procede a incorporarlo a la herramienta. La incorporación se realiza siguiendo las indicaciones establecidas en la sub-sección *Agregar nuevos scripts*. Además, para aprovechar todas las funcionalidades de la herramienta se incluyen todos los campos opcionales, algunos de los cuales se detallan en la sub-sección *Campos relevantes scripts*. Por último, se extiende Nerve para poder distinguir los diferentes *outputs* de los *script*, de acuerdo a lo sugerido en la sub-sección *Determinar existencia de vulnerabilidades*. De este modo, la herramienta es capaz de determinar con certeza cuando el resultado corresponde a una vulnerabilidad.

4.5. Traducción interfaces

La traducción de interfaces a otros idiomas es un problema común en el área de desarrollo web. Originalmente se aborda desarrollando múltiples *templates* en diferentes idiomas. Luego dependiendo de la configuración entregada por el usuario servir el *template* en el idioma adecuado. Sin embargo, este enfoque posee grandes desventajas para implementar y, por lo tanto, no se utiliza mucho hoy en día. En primer lugar, es necesario determinar que partes del código corresponden a texto a traducir y cuales corresponden a código funcional. Además, en el caso que se quiera modificar la traducción de una palabra es necesario buscar todas las apariciones de dicha palabra en todos los archivos, lo que muchas veces no es factible al trabajar con una gran cantidad de archivos. Estos y otros problemas hacen imposible mantener *software* de este estilo.

Actualmente existen tecnologías encargadas de aliviar este proceso y facilitarlo. En Flask existe una librería llamada *Flask-babel* con múltiples funcionalidades relacionadas a la traducción de interfaces, estas permiten adoptar otras metodologías de traducción más inteligentes. En particular, la metodología seleccionada utiliza un solo archivo por lenguaje para almacenar todas las traducciones. Flask luego se encarga de reemplazar el texto de los

templates por las traducciones en el archivo designado. De este modo, basta con solo modificar el archivo de traducciones si se quiere realizar un cambio, lo que facilita en gran medida la mantención del *software* a futuro.

Al utilizar esta configuración ya no es necesario manejar múltiples *templates*, puesto que las librerías se encargan de servir la información en un único *template* utilizando el idioma adecuado. Por otro lado, al centralizar todas las traducciones en un solo archivo al cambiar una de las palabras el cambió se ve reflejado en todas las apariciones de dicha palabra en todos los archivos. Puesto que al servir la información, Flask se encarga de realizar los cambios en todos los archivos que corresponda. Sin embargo, a pesar de estas las ventajas el trabajo sigue siendo considerable. Debido a que aún sigue siendo necesario marcar todos los texto que deben ser traducidos y también traducir el texto propiamente tal. Lo que conlleva una cantidad significativa de tiempo considerando que se tradujeron alrededor de 40 archivos sin contar los *scripts*, los cuales fueron excluidos debido al gran trabajo que significa traducirlos.

Es importante destacar que durante la etapa de marcar los textos relevantes se encontraron múltiples obstáculos. Primero, dependiendo del lenguaje de programación en donde se encuentra contenido el texto es necesario utilizar diferentes formas de marcado, las cuales en muchos casos no son intuitivas y requieren de mayor investigación. Por otro lado, en varias ocasiones no es claro leyendo el código si el texto contenido se muestra en la interfaz o no, por lo que es necesario probar diferentes configuraciones y funcionalidades para determinar que partes del código son relevantes. En consecuencia, para determinar todos las pedazos de código relevantes se realizó un testeo general de todas las funcionalidades de la aplicación.

Durante estas pruebas se encontraron fallas en diferentes módulos de la herramienta, producto de cambios realizados previamente en otras partes del *software*. La mayoría de estos problemas pasaron desapercibidos durante otras etapas del desarrollo debido a que afectan módulos totalmente distintos a los módulos en donde se efectuaron los cambios. Y al momento de desarrollar una funcionalidad es imposible probar cada una de las otras funcionalidades de la herramienta en un tiempo razonable. En general, los mayores problemas encontrados provienen de cambios realizados a funciones compartidas entre módulos, como estructuras de datos o llamadas de estado. Por lo tanto, muchas veces durante el desarrollo de las traducciones fue necesario detenerse y arreglar otros problemas antes de continuar. Lo que finalmente también contribuyó a desarrollar una herramienta robusta.

4.6. Agendado de escaneos

Para lograr la implementación de la funcionalidad es necesario modificar gran parte del funcionamiento base de la herramienta, desde la información desplegada al usuario hasta las condiciones de ejecución de los *daemons*. A continuación, se listan a grandes rasgos los cambios realizados:

1. Mejorar interfaz gráfica para desplegar opciones de escaneo instantáneo, escaneo continuo o agendar escaneo.
2. Guardar configuración de los escaneos en nueva estructura de datos.

3. Ejecutar escaneo al cumplirse la hora agendada.

4. Eliminar información de escaneos previos.

Partiendo por el *frontend*, la herramienta incluye una interfaz que permite configurar los escaneos de acuerdo a las necesidades del usuario. Por lo que basta con extender la interfaz con una opción para agendar los escaneos. En particular, se agrega un botón desplegable para indicar si el escaneo se realizará inmediatamente, se agendará a futuro o si se realizará continuamente en el tiempo. En el caso que se seleccione la opción de agendado, se despliega un calendario interactivo donde el usuario puede seleccionar la fecha y hora correspondiente. La interfaz resultante se muestra a continuación en la Figura 4.5.

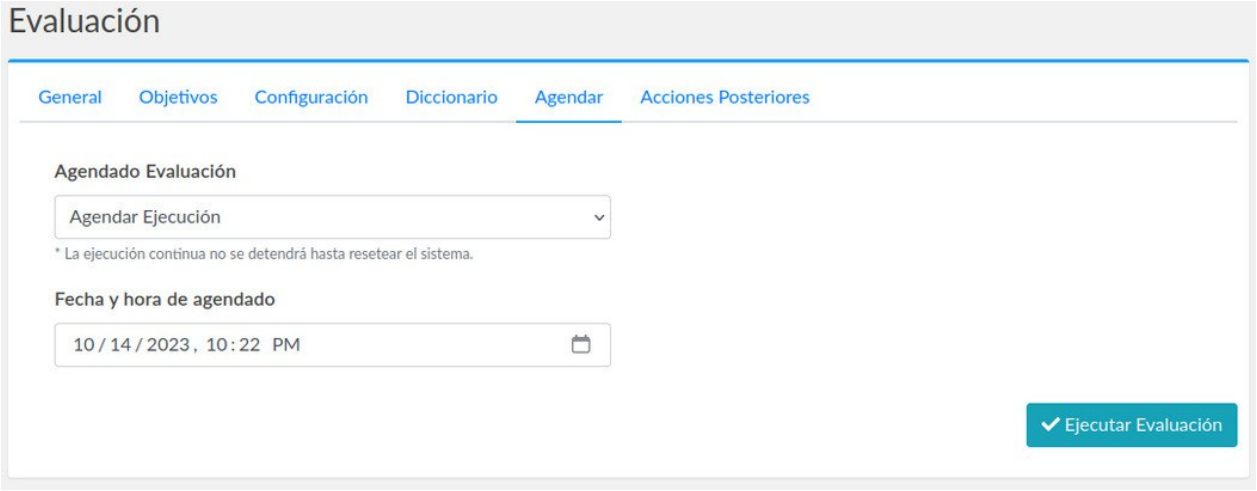

Figura 4.5: Interfaz selección Modo de Escaneo

Al completar el proceso de agendado de escaneos en el *frontend* se envía internamente un campo con la fecha y hora agendada al *backend*, este es usado posteriormente para determinar el momento de ejecución de los *scripts*. El campo es validado tanto por el *frontend* como el *backend* verificando que la información sea una fecha, esté correctamente formateada y corresponda a una fecha razonable en el futuro. En el caso que no se cumpla alguna de estas condiciones no se procede con el escaneo y se despliega un error relacionado al usuario. Al finalizar la validación se continúa con el procesamiento de la información.

Una de las grandes diferencias con la implementación original de la herramienta es la necesidad de guardar la configuración de los escaneos, esto es indispensable para agendar las pruebas a futuro. En contraste con la implementación original la cual inmediatamente empieza el escaneo al obtener la información del *frontend*. Además, es necesario considerar que algunos escaneos tendrán mayor prioridad que otros mientras más cercanos estén a su fecha de ejecución. De ahí que, se busca una estructura de datos simple de operar y que permita almacenar los datos según la fecha.

La estructura que más se acomoda a las necesidades previamente descritas corresponde a un *Ordered Set* de Redis. Un *Ordered Set* permite almacenar información ordenada en función de cierto valor. Por lo tanto, esta estructura permite guardar fácilmente la información de los escaneos de forma ordenada de acuerdo a la fecha en que debe ser

ejecutado. La otra ventaja es que al quitar la información de un escaneo, los escaneos con fecha posterior avanzan una posición en la estructura pero manteniendo el orden. De este modo, basta con acceder al primer valor de la estructura para saber cuando ejecutar el próximo escaneo. Y, al momento de ejecutar un escaneo, es suficiente quitar la configuración de la estructura para que automáticamente el próximo escaneo quede en la primera posición.

El último paso corresponde a ejecutar los *scripts* respetando la fecha de escaneo. El llamado a la ejecución de los *scripts* se encuentra fuertemente ligado a los diferentes *daemons* de la herramienta, por lo que es necesario cambiar las condiciones dentro de estos que gatillan la ejecución de *scripts* para lograr la funcionalidad buscada. El responsable de gatillar la ejecución de los *scripts* es una variable de estado de la herramienta. Por lo que se busca modificar esta principalmente.

Dependiendo del valor de la variable es que se gatillan diferentes procesos. Particularmente, existe un estado que inicia la ejecución de los *scripts*. Este estado se alcanza originalmente cuando un usuario envía la configuración de un escaneo al *backend*. Sin embargo, al introducir la opción para agendar escaneos no hace sentido llamar a la ejecución de *scripts* cada vez que se envían las configuraciones, puesto que la ejecución puede ser en una fecha futura.

A raíz de esto, se cambia la lógica interna que desencadena la ejecución de los escaneos por una condición. En particular se verifica que la fecha del escaneo más próximo sea menor que la fecha del momento. Por lo tanto, solo al alcanzar la fecha de escaneo se inicia la ejecución de los pruebas. Luego, una vez terminado el escaneo, se borra la configuración asociada de la estructura de datos y se avanza en la cola al próximo escaneo. El resto de las funcionalidades post escaneo se mantiene igual, de esta manera, finalizando el nuevo proceso para agendar escaneos.

Capítulo 5

Incorporación de la solución

5.1. Ambiente de control

Antes de introducir la solución en el ambiente de producción se hicieron pruebas en un ambiente de control. Este ambiente es utilizado para probar las funcionalidades desarrolladas en diferentes contextos. De esta forma es posible encontrar errores y problemas provenientes de la etapa de desarrollo, y así arreglarlos antes de incorporar el *software* a producción. Las pruebas también apuntan a garantizar la robustez de la herramienta y asegurar la capacidad de esta para detectar vulnerabilidades en un ambiente poco seguro. Por lo que este ambiente juega un rol fundamental antes de introducir la herramienta en un contexto real.

El ambiente de control utilizado para las pruebas corresponde a una máquina virtual con el sistema operativo Ubuntu 18.04. Se selecciona este sistema operativo puesto que la máquina presente en el ambiente de producción en donde se busca introducir la solución también es de este tipo. La configuración seleccionada para este ambiente considera instalar la herramienta en la máquina y además disponer de otros servicios vulnerables en otros puertos de la misma máquina. De este modo, con una sola máquina es posible utilizar la herramienta y encontrar vulnerabilidades en esta misma.

El principal servicio vulnerable en el entorno es un servidor FTP. Se usa este servicio debido a que el *script* desarrollado se enfoca en una vulnerabilidad de este tipo. El servidor además de presentar la vulnerabilidad del *script* desarrollado dispone de otras vulnerabilidades, como conexión sin credenciales, usuarios con contraseñas débiles y lectura de archivos sensibles. Sumado a esto, la máquina virtual de Ubuntu presenta otras vulnerabilidades como interfaces locales visibles públicamente y puertos que aceptan conexiones externas. Todos estos servicios y configuraciones pueden ser modificados dependiendo de la vulnerabilidad que se desee identificar. En general, son suficientes para llevar a cabo una amplia variedad de pruebas de vulnerabilidades y evaluar diferentes aspectos de la herramienta desarrollada.

Las pruebas corresponden principalmente a escaneos, donde se monitorea el comportamiento de los diferentes módulos de la herramienta. Posteriormente en base a los resultados de las pruebas se corrigen los errores o desviaciones avistadas. Dependiendo del componente de testeo varían los métodos utilizados en las pruebas. Sin embargo,

independiente del método utilizado, durante todas las pruebas se consideran diferentes contextos y configuraciones, para así abordar casos bordes y garantizar la robustez de la herramienta. A continuación se detallan las principales técnicas utilizadas para testear los diferentes componentes de la herramienta.

Partiendo por el *frontend* de la aplicación, se testeo manualmente gran parte de los componentes de las interfaces. Este trabajo involucra verificar que tanto botones como campos, tablas, entre otros, funcionen adecuadamente. Sumado a esto, las pruebas buscan determinar que la información desplegada al usuario es correcta, por lo que también se analiza este aspecto de las interfaces. Al momento de encontrar una falla, se examina el *template* asociado a la falla para determinar el área precisa que gatilla los errores para luego corregir. Por medio de esta metodología es posible preservar la integridad de las interfaces de la herramienta.

En cuanto al testeo de los módulos internos de la herramienta, correspondientes a *backend*,*daemons* y Redis, la metodología se apoya fuertemente en una librería nativa de Flask para logear información de interés y monitorear partes sensibles del código. De este modo, al efectuar una prueba de vulnerabilidad es posible observar el comportamiento de diferentes operaciones y valores internos de la herramienta. La herramienta además incluye una interfaz web para *debugging* en donde se puede visualizar la información *logeada* por la librería.

Aparte de la ejecución de escaneos de manera estándar, utilizando las interfaces de la herramienta, se realizan pruebas introduciendo la información directamente en el *backend* y mediante APIs de la aplicación. Así se garantiza que los módulos internos sean robustos frente a diferentes fuentes de información.

Por último, para garantizar la robustez del *script* desarrollado y su correcta integración en Nerve se utiliza la herramienta Nmap. Nmap incluye funcionalidades para establecer *break points* durante la ejecución de los *scripts* los cuales permiten detenerse en un momento preciso de la ejecución para analizar las diferentes variables y funciones. De este modo identificando potenciales problemas que puedan surgir durante la ejecución. En cuanto a las pruebas efectuadas, se utilizaron diferentes variaciones de parámetros y configuraciones del servidor FTP para probar el *script* en distintos escenarios. De esta manera, el *script* final es robusto frente a diferentes casos bordes encontrados durante las pruebas.

5.2. Instalación ambiente de producción

Luego de terminar el desarrollo de la herramienta y realizar un exhaustivo testeo sobre los diferentes módulos, se procede a incorporar la solución en el ambiente de producción. Esta etapa involucra instalar la herramienta con todas las configuraciones determinadas por la oficina en el ambiente provisto, de modo de dejarla lista para su uso futuro. Se analiza además la forma de acceder a la herramienta y hacer uso de esta.

La máquina en la cual se introduce la solución no posee características distintivas aparte de acceso a la totalidad de la red de la universidad. Similarmente a la máquina del

ambiente de control, esta posee el sistema operativo Ubuntu 18.04. La principal diferencia es que presenta configuraciones más seguras que la otra máquina, pero estas no afectan en gran medida el proceso de instalación. A continuación se detallan los pasos pertenecientes a este proceso.

Para empezar se establece una conexión remota por SSH con la máquina en donde se busca instalar el *software*. Desde ahí se descarga la herramienta clonando el repositorio de Github. Posteriormente se procede a instalar las librerías y herramientas complementarias a la solución. Esto se logra al correr un archivo de configuración que viene con la herramienta, encargado de analizar el tipo de *software* y versiones de sistema para instalar las dependencias adecuadas. Luego de descargar todos los archivos se sigue adelante con la configuración de las variables y servicios complementarios. Esto aborda configurar la base de datos, valores por defecto de los escaneos y *hooks* externos, como correo electrónico, de acuerdo a las preferencias de la OSI. Al finalizar con las configuraciones la herramienta se encuentra lista para su uso.

La instalación previamente mencionada sirve para hacer uso de la herramienta directamente desde la máquina en la que se encuentra instalada. Pero para hacer uso remota de esta aún faltan algunas configuraciones locales que considerar. Principalmente falta transmitir la información de las interfaces de la herramienta (máquina remota) hasta la máquina donde se inicializa la conexión SSH (máquina local). La solución involucra establecer un túnel entre ambas máquinas para traspasar la información con un *proxy* intermedio.

SSH facilita este proceso mediante *dynamic port fowarding*, esta configuración permite detectar el puerto de la máquina remota en la que se despliegan las interfaces de usuario y abrir un servidor *proxy* SOCKS 5 en la máquina local [27]. De este modo se establece una conexión entre ambas máquinas con el *proxy* local como mediador. El servidor *proxy* permite controlar el tráfico que se envía desde la máquina local a la remota y viceversa. Por lo tanto, al conectar un navegar web local al servidor *proxy* es posible visualizar la información enviada por la máquina remota, en este caso las interfaces de la herramienta Nerve, pero localmente. Posteriormente, el usuario interactúa con las interfaces desplegadas en el navegador y se envían las solicitudes al *proxy* quien a su vez las envía a la máquina remota. La máquina remota interpreta estas solicitudes y envía las respuestas devuelta al *proxy*, repitiendo el ciclo de comunicación entre las máquinas y simulando una interacción directa con la herramienta.

5.3. Pruebas en red de la universidad

Una vez instalada la herramienta y establecida la conexión remota con éxito, se procede a probar la herramienta en la red de la universidad. Estas pruebas apuntan a verificar que la herramienta es capaz de detectar vulnerabilidades a gran escala en la red. Como objetivo de las pruebas se dispuso de 5 sub-redes diferentes de distintas entidades universitarias, correspondientes al Centro de Computación(CEC), Departamente de Ciencias de la Computación(DCC), VTI, Observatorio Cerro Calan y algunos sistemas presentes en Torre 15. Previo a las pruebas se consultó con los encargados correspondientes y se establecieron ciertas restricciones.

En el caso de las redes CEC/DCC/VTI se tomaron varios precauciones durante las pruebas. Principalmente para no afectar el desempeño de las máquinas atacadas y tampoco saturar la red. Por lo tanto, las pruebas en estas redes solo se efectuaron en el puerto 22 (SSH) de los sistemas. Sumado a esto, se quitaron los scripts de fuerza bruta y denegación de servicio(DoS), reduciendo en gran medida los scripts totales a ejecutar.

En consecuencia, aparte de los scripts de descubrimiento y versionamiento, las pruebas se limitan a ejecutar 5 scripts principales. Los cuales apuntan a identificar los tipos de conexiones que acepta el servicio SSH, determinar si el servicio muestra algún *path* interno de la máquina, identificar alguna versión antigua presente y buscar una vulnerabilidad del estilo Log4Shell [28]. Del total de *scripts*, 2 corresponden a *scripts* NSE que fueron agregados a la herramienta durante este trabajo. Uno de ellos pertenece al repositorio de Nmap y el otro fue sacado de la comunidad *open source*. Todos estos *scripts* apuntan a encontrar vulnerabilidades SSH en las redes sin utilizar ataques de alto impacto.

En el caso de las redes de Torre 15 y Cerro Calan se dieron mayores libertados para las pruebas. Principalmente se permitió analizar todos los puertos de las máquinas presentes en las redes. Por ende, la cantidad de *scripts* utilizados para estas pruebas es mucho mayor. Sin embargo, las restricciones de ataques de fuerza bruta, denegación de servicio(DoS) y *scripts* de alto impacto también se aplicaron para este caso. En total los *scripts* utilizados para las pruebas ascienden a alrededor de 50.

5.4. Resultados obtenidos

5.4.1. Resultados redes CEC, DCC y VTI

Los resultados de las pruebas efectuadas en las redes CEC/DCC/VTI arrojaron 167 *hosts* de los cuales 79 presentan servicios SSH activos. En la sección B.1 Resultados CEC, DCC y VTI del anexo se muestran algunas de las interfaces de la herramienta con los resultados asociados a estas pruebas. Cabe destacar, que para efectos de la herramienta la detección de cada uno estos servicios SSH corresponde a una vulnerabilidad. La Figura 5.1 muestra el detalle asociado.

TÍTULO	This rule checks for open Remote Management Ports
RESULTADOS	Remote Server Exposes Administration Port(s)
DIRECCIONES	[redacted]
PUERTOS	22
DETALLES	Server is listening on remote port: 22 (SSH)
ID_REGLAS	SVC_6509
MITIGACIÓN	Bind all possible services to localhost, and confirm only those which require remote clients are allowed remotely.

Figura 5.1: Detalle vulnerabilidad: Servicio SSH encontrado.

La razón por la que esta información corresponde a una vulnerabilidad se debe a que esta información al ser pública incentiva a atacantes a realizar ataques posteriores más especializados. Y una forma de lidiar con estas amenazas es esconder esta información y solo permitir conexiones específicas con permisos previamente establecidos. Uno de los mecanismos que permite este tipo de configuración se llama *Port Knocking* [29]. Este

mantiene el puerto del servicio cerrado y solo lo abre cuando un usuario autorizado exige conectarse. Para lograr esto se utiliza un *firewall* encargado de habilitar el servicio cuando un usuario solicita la correcta secuencia de puertos. Otra opción común, pero menos segura, es correr el servicio en otro puerto diferente al por defecto para engañar a adversarios. Sin embargo, ocultar la existencia de un servicio siempre será una medida de seguridad inferior a aumentar la seguridad del servicio propiamente tal.

Por otro lado, como se puede ver en la Figura 5.2, se detectaron algunos servicio SSH que aceptan contraseñas como método de autenticación. Esto tiende a ser menos seguro que las conexiones con llaves públicas y privadas. Las contraseñas son más propensas a ataques de fuerza bruta por lo que pueden presentar un mayor riesgo en algunos casos. En consecuencia, este tipo de vulnerabilidades busca dar conocimiento de que los sistemas son propensos a este tipo de ataques.

TÍTULO	This rule checks if OpenSSH allows passwords as an accepted authentication mechanism
RESULTADOS	Remote Server Supports SSH Passwords
DIRECCIONES	[redacted]
PUERTOS	22
DETALLES	Server accepts passwords as an authentication option
ID_REGLAS	CFG_FOQW
MITIGACIÓN	OpenSSH allows password authentication, this is considered bad security practice. SSH Key based authentication should be enabled on the server, and passwords should be disabled.

Figura 5.2: Detalle vulnerabilidad: SSH acepta contraseñas.

En general, los resultados obtenidos evidencian pocas vulnerabilidades en las redes escaneadas. Pese a que la herramienta muestra un gran número de vulnerabilidades, asociado a la gran cantidad de servicios SSH, estas no necesariamente se traduce en un gran número de fallas de seguridad directamente. Muchos de los sistemas de la universidad son públicos para los estudiantes y académicos, como tal deben estar disponibles para conexiones externas. Por lo que puede que muchos sistemas con servicios SSH visibles al exterior deban ser descartados, reduciendo en gran medida el número total de vulnerabilidades. Para los encargados de los sistemas estos resultados son buenos, ya que no divisan vulnerabilidades severas como versiones antiguas del *software*, *exploits* tipo log4shell ni filtrado de *paths* internos de las máquinas.

El bajo número de vulnerabilidades se atribuye a 2 causas principales. En primer lugar, las restricciones impuestas en las pruebas limitan en gran medida el número de *scripts* ejecutados y, por lo tanto, el número de vulnerabilidades a encontrar. En segundo lugar, los servicios analizados pertenecen a organizaciones que constantemente realizan chequeos y mantienen al día sus servicios. Por lo que es difícil encontrar vulnerabilidades del estilo *low-hanging fruit* en estas circunstancias. En suma, las restricciones de las pruebas y medidas de seguridad de los sistemas dificultan la búsqueda de vulnerabilidades en la red.

5.4.2. Resultados redes Torreo 15 y Cerro Calan

Las pruebas efectuadas en las redes del Cerro Calan y Torre 15 presentan resultados diferentes a los de las otras redes. El detalle se muestra en la sección B.2 del anexo. En el

caso del Cerro Calan se detectaron 70 vulnerabilidades distribuidas a lo largo de 22 *hosts*. Y en el caso de Torre 15, se encontraron un total de 128 vulnerabilidades. La gran cantidad de puertos escaneados en estas pruebas contribuye mayoritariamente a este gran número.

En general, la herramienta detectó una gran cantidad de servicios en la red como RTSP, FTP, NFS, etc. Similar a las vulnerabilidades asociadas a los servidores SSH en las redes CEC/DCC/VTI, estos servicios también son marcados como vulnerables. Puesto que la herramienta estima que esta información debería estar escondida de atacantes externos. Y del total de vulnerabilidades encontradas la gran mayoría pertenece a esta categoría.

Dentro de los servicios web se detectó además una cantidad considerable de sistemas que aceptan autenticación básica como medida de autenticación. El detalla de la vulnerabilidad se muestra en la Figura 5.3. Esta vulnerabilidad es particularmente peligrosa puesto que hoy en día esta configuración es vulnerable a ataques de fuerza bruta y puede ser explotada por atacantes. Sumado a esto, se detectaron un par de páginas web que aceptan formularios sin encriptación, referirse a Figura 5.4. Esta vulnerabilidad permite a atacantes interceptar los mensajes y leerlos. Por lo que ambas vulnerabilidades presentan serios riesgos para los sistemas involucrados.

TÍTULO	This rule checks if a Web Server has Basic Authentication enabled
RESULTADOS	Basic Authentication is Configured
DIRECCIONES	[redacted]
PUERTOS	80
DETALLES	Basic Authentication is Configured at http://172.27.96.242:80/
ID_REGLAS	CFG_D2A9
MITIGACIÓN	Basic authentication is a simple authentication scheme built into the HTTP protocol. The client sends HTTP requests with the Authorization header that contains the word Basic word followed by a space and a base64-encoded string username:password Basic Authentication does not have brute force protection mechanisms, and may potentially be a target for attackers

Figura 5.3: Detalle vulnerabilidad: Servicio con configuración básica habilitada.

TÍTULO	This rule checks for password forms over HTTP protocols
RESULTADOS	Unencrypted Login Form
DIRECCIONES	[redacted]
PUERTOS	80
DETALLES	Login Page over HTTP at http://172.27.96.232:80/
ID_REGLAS	VLN_SKKF
MITIGACIÓN	Website accepts credentials via HTML Forms, howeverm, it offers no encryptions and may allow attackers to intercept them.

Figura 5.4: Detalle vulnerabilidad: Página web acepta formularios sin encriptación.

En definitiva, los resultados de las pruebas en estas redes son más satisfactorios que los obtenidos previamente, puesto que sacan a luz algunas configuraciones más graves de los sistemas. La principal diferencia con las otras pruebas, se debe a la mayor libertad en las configuraciones, lo que ayuda a vislumbrar y detectar estos problemas. También los

sistemas de estas redes tienden a ser más vulnerables en promedio, debido a que no son tan regularmente mantenidos como el resto. De ahí la importancia de una herramienta como esta para encontrar estas vulnerabilidades antes que atacantes externos puedan explotarlas.

5.5. Traspaso de la herramienta

5.5.1. Capacitación

La última parte del trabajo involucra capacitar al personal en la instalación, uso y mantención de la herramienta. Por consiguiente, se efectuaron un par de sesiones con el personal de la oficina en donde se mostró la herramienta y se explicó su funcionamiento. Aquí se abordaron diferentes aspectos desde como agregar nuevos *scripts* hasta como actualizar la herramienta con las últimas versiones disponibles.

Al ser una herramienta fácil de utilizar y al tener experiencia previa con herramientas similares, la OSI no tuvo mayores problemas para aprender las aspectos principales de la herramienta. La mayor dificultad encontrada tuvo relación con como agregar nuevos *scripts* a la herramienta. Dado que es necesario interactuar con varios módulos de la solución si se quiere sacar el mayor provecho. Pero en general la capacitación y traspaso de la herramienta pudieron realizarse sin problemas.

5.5.2. Documentación

Con el fin de facilitar el aprendizaje y uso de la herramienta se incluyen algunos documentos de ayuda. Los documentos apuntan a guiar a los usuarios en diferentes casos de uso y también explicar diferentes componentes internos de la herramienta por si se quiere extender a futuro. Esta información se encuentra disponible en el README del repositorio y en la interfaz *Documentación* de la herramienta misma.

A continuación se listan las temas principales contenidos en estos documentos con una pequeña descripción. Más información al respecto puede encontrarse en el anexo C.

- Guía de instalación: Instrucciones para instalar la herramienta en el ambiente de producción de la OSI y como iniciar/terminar sus servicios asociados.
- Forma de uso de la herramienta: Corresponde a información para instruir al usuario en el manejo de la herramienta.
- Guía de incorporación de *scripts* en la herramienta: Instrucciones para introducir nuevos *scripts* en la herramienta y ejecutarlos en futuras pruebas.
- Documentación para desarrolladores: Información interna como arquitectura del *software*, detalle de componentes, documentación código y flujo entre distintos módulos.

Capítulo 6

Conclusiones y trabajo futuro

En el siguiente capítulo se presentan las conclusiones del trabajo desarrollado. Se incluye un recuento de los objetivos cumplidos y un resumen de las principales etapas del trabajo. También se recalcan los aspectos que podrían haberse hecho mejor y otras facultades que podrían agregarse a la herramienta.

Primero destacar que la solución resultante aborda todos los requisitos propuestos en un comienzo. Se logra extender una herramienta preexistente para adecuarse a los problemas de la oficina e incorporarla en el ambiente de producción. Esta herramienta facilita la búsqueda de vulnerabilidades en la red de la universidad y es capaz de automatizar el proceso de ejecución de *scripts*, el cual se ejecutaba manualmente previo a este trabajo. Sumado a esto, la herramienta presenta resultados satisfactorios al realizar pruebas de vulnerabilidades en el ambiente de producción.

Por su parte la OSI se encuentra satisfecha con la solución. La capacidad para agregar *scripts* de la herramienta, el bajo costo asociado, el bajo impacto en la red y la facilidad de uso son todas cualidades bien valoradas por la organización. Los resultados de las pruebas evidencian además el potencial de la herramienta al utilizarla en un ambiente adecuado. Considerando todo, la solución ayuda a mantener la seguridad de la red de la universidad y es simple de manejar.

A continuación, se resumen las etapas más importantes del trabajo para alcanzar la solución final desde un punto de vista más crítico. El trabajo inicia definiendo los requisitos y alcances del trabajo en conjunto con la OSI. Se acuerda utilizar una herramienta preexistente en vez de desarrollar una desde cero dada la existencia de herramientas ya disponibles que permiten abordar las problemáticas planteadas. Lo que da paso a investigar la herramienta que mejor se acomoda a la situación de la oficina considerando sus diferentes características.

Durante esta etapa se aprende sobre los diferentes tipos de escáners y las funcionalidades que los distinguen. En la búsqueda de la mejor opción se consideran las diferentes limitaciones del contexto en donde se introduce la solución para finalmente seleccionar la herramienta *Nerve*. Este proceso fue crítico para entender en detalle el problema a resolver y las cualidades de la solución a desarrollar.

Posterior a seleccionar la herramienta y explorar los diferentes ámbitos que esta aborda, se continúa a desarrollar y extender la solución escogida. Se procede con este enfoque debido a que herramienta escogida no permite abordar todos los requisitos establecidos por la oficina de forma nativa. A pesar de esto, es posible aprovecharse de la característica *open source* de la herramienta para extenderla y así abordar todos los requisitos. Gran parte del tiempo total de trabajo se dedicó a esta etapa, ya sea a entender el código preexistente de la herramienta o en el desarrollo de las funcionalidades propiamente tal. El proceso destaca por la gran variedad de problemas abordados, desde el desarrollo de funcionalidades meramente estéticas, como interfaces web, hasta alterar el comportamiento interno de la herramienta para permitir agendar pruebas de vulnerabilidades. Esta gran variedad también contribuye a aprender sobre diferentes temas como el lenguaje de programación Lua, servicios FTP y estructuras de datos en Redis.

Posteriormente cada una de las funcionalidades y requisitos desarrollados se sometieron a pruebas en un ambiente de control para detectar potenciales problemas y solucionarlos. Este ambiente apunta a verificar que tanto la herramienta como el *script* desarrollado funcionan adecuadamente. El proceso es clave para desarrollar una herramienta robusta y confiable, lo que es especialmente importante cuando se trata de un escáner de vulnerabilidades.

La última etapa del trabajo involucra probar la herramienta en la red de la universidad. En este caso se obtuvieron resultados mixtos. Por un lado, las pruebas en las redes CEC/DCC/VTI no vislumbran vulnerabilidades muy relevantes. Esto se debe en parte a las restricciones a las que fueron sometidas las pruebas y a la mayor seguridad de los servicios atacados. Por otro lado, las redes del Cerro Calan y Torre 15 dejan en evidencia algunas problemas de configuraciones que pueden ser potencialmente bastante peligrosos. Aquí brilla este tipo de herramienta por su capacidad de analizar grandes redes (sin mayor impacto) y su capacidad para detectar vulnerabilidades específicas en sistemas poco mantenidos. De esta manera, los resultados reafirman la utilidad de la herramienta en un ambiente real y permiten a la herramienta posicionarse como una sólida alternativa de detección de vulnerabilidades en la red.

Cabe mencionar que a pesar de cumplir con el principal objetivo del trabajo y presentar buenos resultados en las pruebas, la herramienta presenta algunas desventajas necesarias a considerar al momento de utilizarla. En primer lugar, no hay que olvidar que es una herramienta enfocada en vulnerabilidades del estilo *low-hanging fruit*, en consecuencia, no es suficiente el uso de este escáner para garantizar la seguridad total de la red. Dependiendo del tipo de máquinas y servicios presentes en la red puede ser requerido al menos 1 escáner adicional. En el caso de la red de la universidad la gran cantidad de dispositivos y variedad de servicios requiere de múltiples escáners para brindar una buena cobertura. Por lo que si bien este proyecto corresponde a un avance en términos de seguridad sigue siendo insuficiente para garantizar la seguridad de la red en todos sus ámbitos.

Por otro lado, hay que ser crítico en algunos aspectos de la solución. Dadas las restricciones de tiempo del trabajo ciertos requisitos desarrollados no se abordaron de la mejor forma posible. El principal ejemplo se asocia al requerimiento de introducir nuevos *scripts* en la herramienta. En vez de desarrollar una nueva interfaz para subir nuevos *scripts*

se opta porque el usuario introduzca directamente el archivo en la carpeta de *scripts*. Esta metodología es mucho más propensa a fallas por parte del usuario pero posee la ventaja de que es más fácil de implementar y permite ahorrar una cantidad considerable de tiempo. De igual manera, hay otros requisitos que también podrían haberse implementado de mejor forma al disponer de más tiempo.

Asimismo, hay varias modificaciones o extensiones que se pueden realizar para agregar valor a la solución final. A continuación, se proponen algunas de estas ideas. Partiendo por introducir más *scripts* a la herramienta, ya que una de las principales críticas en relación a otras herramientas es la poca cantidad de *scripts* presentes en la herramienta base y que estos no son actualizados regularmente.

Durante el trabajo se opta por agregar opciones para que los usuarios pueden introducir sus propios *scripts*. Mientras esto representa un avance siguen habiendo complicaciones como la necesidad de introducir y mantener los *scripts* manualmente. Bajo estas circunstancias, puede ser interesante bases de datos de vulnerabilidades públicas. Estas generalmente son mantenidas por organizaciones y presentan una gran gama de vulnerabilidades, incluyendo las últimas descubiertas. Sin embargo, la integración de estas bases de datos trae consigo grandes dificultades para implementar debido a los distintos formatos de los *scripts* y la necesidad de ir actualizando los datos periódicamente. Aún así, dependiendo de las necesidades de la oficina puede que sea factible de implementar a futuro.

Otra funcionalidad que puede ser de gran utilidad es la capacidad de formar grupos de *scripts* en la herramienta. De momento solo se da la opción de agrupar los *scripts* de acuerdo a su *Nivel de Agresividad* durante la ejecución. No obstante, uno de los operadores expresó interés en la capacidad de formar grupos personalizados. De este modo, el operador puede crear grupos de *scripts* de acuerdo a normas chilenas de seguridad y efectuar pruebas para verificar su cumplimiento en un solo escaneo. En consecuencia, dependiendo del uso de la herramienta esta funcionalidad puede ser de gran utilidad.

Por otro lado, un aspecto que distingue negativamente a la herramienta de otras alternativas es la forma en que realiza sus escaneos. Como se mencionó previamente en la introducción de la herramienta Nerve, la herramienta al momento de efectuar pruebas de vulnerabilidades ejecuta todos sus *scripts*, independiente del servicio atacado. Por lo tanto, muchos de los *scripts* son ejecutados en vano al no coincidir el tipo de un servicio con el tipo de los *scripts*. Este desperdicio de recursos puede evitarse al verificar de antemano los tipos de los objetivos y ejecutando *scripts* acorde a estos. La herramienta ya realiza este chequeo pero aún falta implementar la lógica para filtrar los *scripts*. Esto podría mejorar aún más el desempeño de la herramienta.

Como reflexión final, destacar que la solución se acomoda a las necesidades actuales de la OSI pero puede que a futuro estas cambien y sea necesario enfocarse en nuevos objetivos. En tal caso la facilidad para extender la herramienta con la documentación preexistente ayudan a continuar con el desarrollo y moldear la solución de acuerdo a las nuevas necesidades. Teniendo esto en mente, junto con los trabajos futuros recomendados, se entrega el presente trabajo a la oficina para su uso y continuo desarrollo.

Bibliografía

[1] Delaere, C. *et al.*, "Reporte ciberseguridad 2022 y tendencias 2023 en chile y latinoamérica," 2023, https://entelocean.com/reporte-ciberseguridad (visitado el 26-06-2023).

[2] Inda, I. *et al.*, "Informe anual de gestión csirt 2022," 2023, https://csirt.gob.cl/estadisticas/informe-anual-de-gestion-csirt-2022/ (visitado el 26-06-2023).

[3] Paytm, "Nerve," 2020, https://github.com/PaytmLabs/nerve.

[4] Greenbone, "Greenbone opensource," 2023, https://www.greenbone.net/en/open-source-vulnerability-management/.

[5] Nmap, "Nmap scripting engine (nse)," 2007, https://nmap.org/book/nse.html (visitado el 26-06-2023).

[6] Ierusalimschy, R., Celes, W., y de Figueiredo, L. H., "Lua," 2023, https://www.lua.org/.

[7] Nmap, "Nmap," 2023, https://nmap.org/.

[8] Basu, S., "Types of vulnerability scanning," 2023, https://www.getastra.com/blog/security-audit/vulnerability-scanning-types/ (visitado el 26-06-2023).

[9] Balbix, "Vulnerability scanners and scanning tools: What to know," 2023, https://www.balbix.com/insights/what-to-know-about-vulnerability-scanning-and-tools/ (visitado el 26-06-2023).

[10] Fernandez, R., "What is vulnerability scanning & how does it work?," 2023, https://www.esecurityplanet.com/networks/vulnerability-scanning-what-it-is-and-how-to-do-it-right/ (visitado el 26-06-2023).

[11] Synk, "Vulnerability scanner: what is it and how does it work?," 2023, https://snyk.io/learn/vulnerability-scanner/ (visitado el 26-06-2023).

[12] Crowdstrike, "What is privilege escalation?," 2022, https://www.crowdstrike.com/cybersecurity-101/privilege-escalation/ (visitado el 26-06-2023).

[13] Scarfone, K., Souppaya, M., y Amanda Cody, A. O., "Technical guide to information security testing and assessment," NIST, pp. 6–2, 2008, doi:https://doi.org/10.6028/NIST.SP.800-115.

[14] News, T. H., "Vulnerability scanning frequency best practices," 2021, https://thehackernews.com/2021/12/vulnerability-scanning-frequency-best.html (visitado el 26-06-2023).

[15] Fortra, "Tripwire ip360," 2023, https://www.tripwire.com/products/tripwire-ip360 (visitado el 26-06-2023).

[16] McGovern, R., Curtin, E., y Voicilas, R., "inotifywait," 2022, https://www.man7.org/linux/man-pages/man1/inotifywait.1.html.

[17] Inc, T., "Nessus," 2023, https://www.tenable.com/products/nessus (visitado el 26-06-2023).

[18] Rogers, R., "Chapter 11 - nasl," en Nessus Network Auditing (Second Edition) (Rogers, R., ed.), pp. 331–364, Burlington: Syngress, second edition ed., 2008, doi:https://doi.org/10.1016/B978-1-59749-208-9.00011-3.

[19] Rapid7, "Nexpose," 2023, https://www.rapid7.com/products/nexpose/ (visitado el 26-06-2023).

[20] Rapid7, "Nexpose writing vulnerability checks," 2023, https://docs.rapid7.com/insightvm/writing-vulnerability-checks/ (visitado el 26-06-2023).

[21] Inouye, J., "What is markup language," 2022, https://hackr.io/blog/what-is-markup-language (visitado el 26-06-2023).

[22] Rapid7, "Nexpose features," 2023, https://www.rapid7.com/products/nexpose/features (visitado el 26-06-2023).

[23] Foundation, P. S., "Python," 2023, https://www.python.org/.

[24] Ronacher, A., "Flask," 2020, https://pypi.org/project/Flask/.

[25] Sanfilippo, S., "Redis," 2023, https://redis.io/.

[26] Norman, A., "Python-nmap," 2021, https://pypi.org/project/python-nmap/.

[27] Leech, M. *et al.*, "Socks protocol version 5," RFC 1928, RFC Editor, 1996.

[28] CISA, "Mitigating log4shell and other log4j-related vulnerabilities," 2021, https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a (visitado el 26-06-2023).

[29] Krzywinski, M., "Port knocking: Network authentication across closed ports," SysAdmin Magazine, vol. 12, pp. 12–17, 2003.

[30] Goodger, D., "restructuredtext," 2002, https://peps.python.org/pep-0287/ (visitado el 26-06-2023).

[31] Goodger, D., "Pep 287 – restructuredtext docstring format," 2002, https://peps.python.org/pep-0287/ (visitado el 26-06-2023).

Anexo A

Listado Requisitos

Tabla A.1: Listado completo de requisitos

Categoría	Requisito	Ya viene implementado
Automatización	Capacidad para ejecutar *scripts* de la oficina en el formato definido	X
Automatización	Permitir procesar información de los resultados	–
Automatización	Desplegar resultados obtenidos en *dashboard*	–
Automatización	Generar reportes a partir de pruebas realizadas	–
Automatización	Opción para enviar reporte a encargados	✓
Automatización	Interfaces para manejar el proceso de automatización	✓
Incorporación *scripts*	Seleccionar formato estándar al cual transformar *scripts* OSI	✓
Incorporación *scripts*	Formato debe ser fácil de extender, completo y "popular"	✓
Incorporación *scripts*	Transformar 1 *script* al formato	X
Incorporación *scripts*	Al transformar *scripts* asegurarse que sean robustos	X
Incorporación *scripts*	Herramienta permita incorporar nuevos *scripts* (extensible)	–
Incorporación *scripts*	Capacidad para entregar parámetros de los *scripts* a la herramienta	X
Facilidad de uso	Opción para realizar escaneos rápidos o a partir de mínimas configuraciones	✓
Facilidad de uso	Interfaces fáciles de acceder y navegar	✓
Facilidad de uso	Interfaces en español	X
Complementarios	Opciones que permitan realizar escaneos más especializados y personalizables	–

Complementarios	Agendado de escaneos	X
Complementarios	Autenticación en pruebas de vulnerabilidades	X
Complementarios	Nivel de severidad asociado a vulnerabilidades encontradas	–
Complementarios	Herramienta robusta en general	✓
Complementarios	Pruebas de vulnerabilidades de poco impacto en la red de la universidad	✓
Entrega de la solución	Instalar correctamente la herramienta en el ambiente OSI	X
Entrega de la solución	Probar *scripts* en red de la universidad	X
Entrega de la solución	Capacitación de personal en uso y mantención de la herramienta	X
Entrega de la solución	Incluir documentación necesaria para manejar todos los aspectos de la herramienta	X

La columna *Ya viene implementado* de la tabla determina si el requisito se encuentra implementado previo a la etapa de desarrollo del trabajo. Puesto que algunos de estos requisitos ya vienen implementados en la herramienta seleccionada. Recalcar que durante la etapa de desarrollo si se aborda el resto de los requisitos.

El símbolo usado para mostrar que el requisito **no** ha sido desarrollado previo a la etapa de desarrollo es "X". Por otro lado "–" se usa cuando el requisito en cuestión solo ha sido abordado parcialmente previo a esta etapa. Por último, el símbolo "✓" es utilizado si el requisito ya se encuentra implementado.

Anexo B

Resultados pruebas red universidad

B.1. Redes CEC, DCC y VTI

Figura B.1: *Dashboard* herramienta luego de escanear red VTI.

Figura B.2: *Dashboard* herramienta luego de escanear redes CEC/DCC.

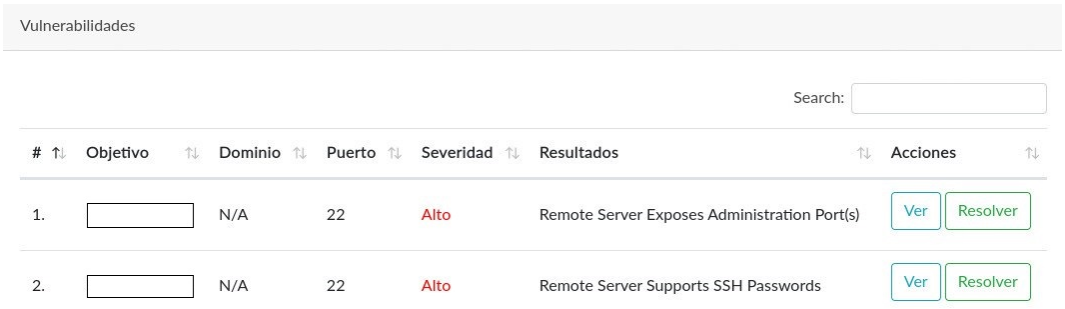

Figura B.3: Principales vulnerabilidades redes CEC/DCC/VTI.

B.2. Redes Cerro Calan y Torre 15

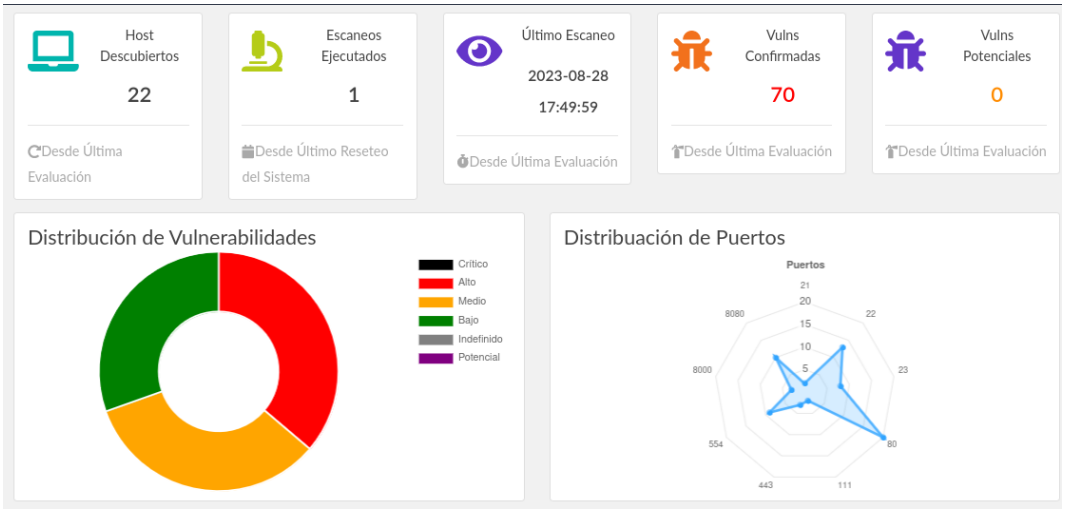

Figura B.4: *Dashboard* herramienta luego de escanear red Cerro Calan.

Figura B.5: *Dashboard* herramienta luego de escanear red Torre 15.

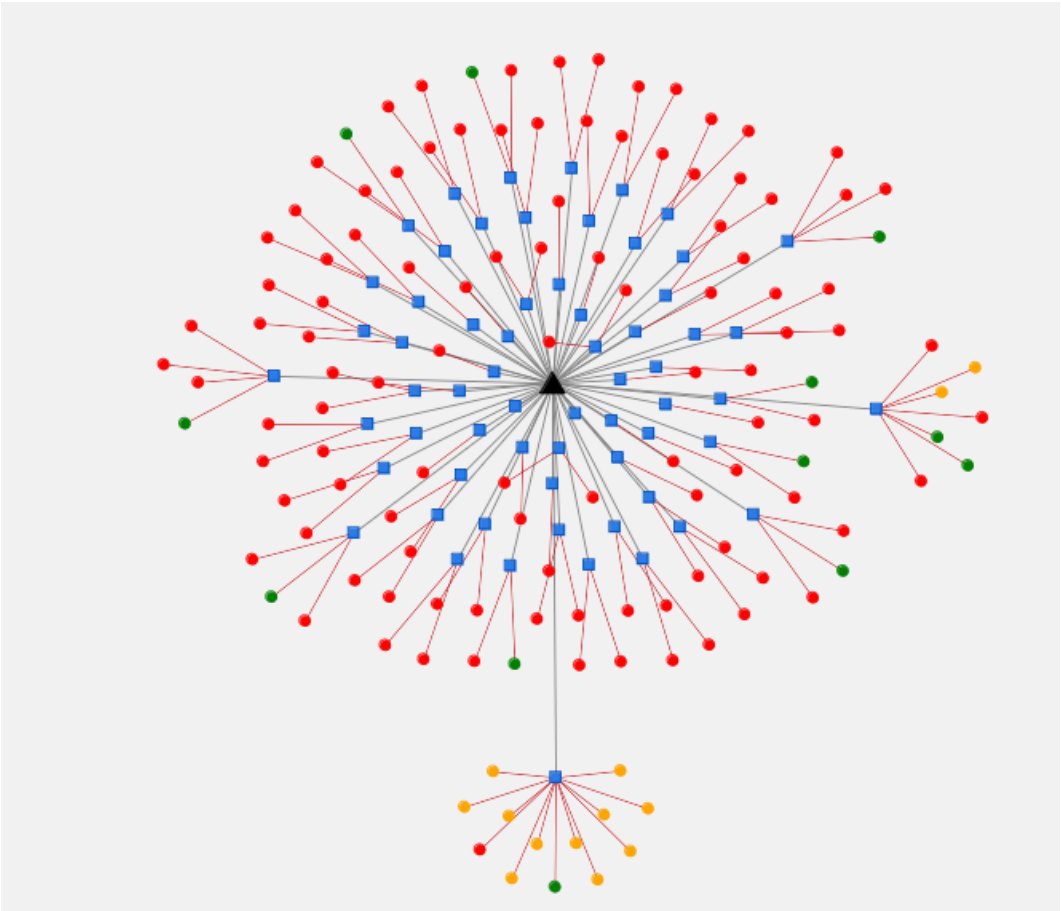

Figura B.6: Topología red Torre 15.

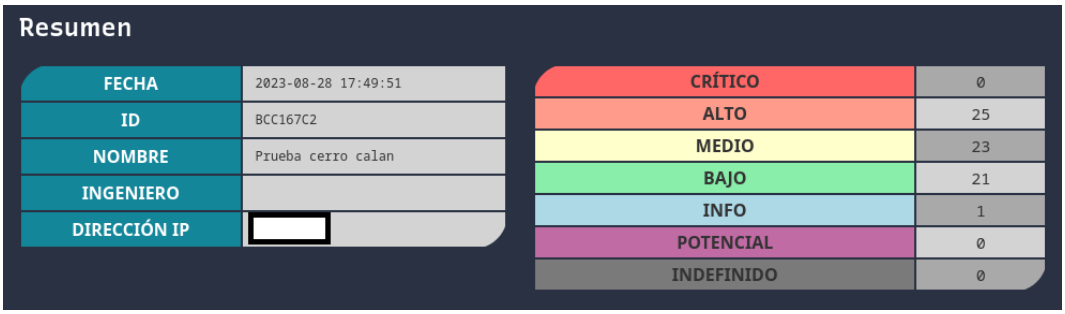

Figura B.7: Resumen vulnerabilidades Cerro Calan.

Anexo C

Documentación Nerve

C.1. Instalación

Prerequisitos

Nerve instalará todos los prerequisitos automaticamente al escoger la opción de instalación en el servidor(Testeado en Ubuntu 18.x)(al correr el script `install/setup.sh`). El proyecto original también viene con un Dockerfile para su conviencia.

Es importante mencionar que Nerve requiere de acceso *root* para la configuración inicial en la máquina(instalación de paquetes, etc).

Servicios y Paquetes requeridos para que Nerve pueda correr:

- Servidor Web (Flask)
- Servidor Redis (local)
- Paquete Nmap (Binario y librería de Python llamda `python-nmap`)
- Acceso a conexiones entrantes en el puerto HTTP/S (esto se puede definir en config.py)

El *script* de instalación se encarga de todo, pero si se opta por una instalación manual es necesario considerar estos requerimientos.

Visualización de interfaces remotas

Para manejar remotamente la interfaces de la herramienta es necesario configurar un tunel que permita interactuar con las interfaces remotamente. La forma más simple de lograr esto es mediante una conexión SSH y un servidor *proxy* local conectado al navegador web de preferencia. A modo de ejemplo se listan los los pasos utilizando el navegador web firefox:

1. Establecer conexión SSH con la máquina en donde se aloja la herramienta y levantar servidor *proxy* local en puerto 8888: `ssh -D localhost:8888 usuario@nerveIP`
2. Configurar firefox con el servidor *proxy*: Configuraciones Firefox -> Proxy -> Socks 5 host:localhost:8888
3. Visualizar interfaz: http://IPMaquinaRemota:PuertoMaquinaRemota

Instalación Server

Navegar a /opt

```
cd /opt/
```

Clonar el repositorio

```
git clone git@github.com:TomasTorresB/nerve.git && cd nerve
```

Correr el instalador (requiere root)

```
bash install/setup.sh
```

Chequear que NERVE corra

```
systemctl status nerve
```

En el navegador web, visitar http://ip.add.re.ss:8080 y utilizar las credenciales imprimidas en el terminal.

Instalación Multi Nodo

En el caso que se prefiera una instalación multi-nodo de la herramienta, se pueden seguir las intrucciones básicas de instlación y luego:

1. Modificar el archivo config.py en cada nodo
2. Cambiar el "server address" de Redis a `RDS_HOST` para que apunte a servidor central de Redis al que todas las instacias de Nerve reportarán.
3. Correr `service nerve restart` o `systemctl restart nerve` para recargar las configuraciones
4. Correr `apt-get remove redis` / `yum remove redis` (Dependiendo de la distribución de Linux) dado que no sera necesario una instancia para cada nodo. No olvidar permitir al puerto 3769 recibir conexiones entrantes en la instancia de Redis, de modo que las instancias de Nerve puedan comunicarse con la base de datos.

Upgrade

En el caso de querer mejorar la plataforma, lo más fácil es simplemente clonar nuevamente el repositorio nuevamente el repositorio y sobreescribir todos los archivos manteniendo los archivos claves como configuraciones. Los pasos se listan a continuación:

- Hacer una copia del archivo `config.py` en el caso de querer guardar las configuraciones.
- Borrar `/opt/nerve` y nuevamente hacer git clone.
- Mover el archivo `config.py` devuelta a `/opt/nerve`.
- Reanudar el servicio utilizando `systemctl restart nerve`.

Se puede configurar un *cron task* para realizar mejorar automáticas de Nerve. Hay un API *endpoint* que permite checkear las últimas versiones disponibles que se puede utilizadar para estos propositos: `GET /api/update/platform`

C.2. Forma de uso

Inicio Rápido

Inicio Rápido es la mejor manera de probar NERVE en su forma más simple. Es similar a correr un escaneo básico sin la necesidad de especificar como ejecutarlo.

Al utilizar esta opción es requerido definir la notación CIDR de la red, Inicio Rápido **no soporta DNS.**

Por ejemplo, en el caso de querer escanear su propia red loca, basta con ingresar **192.168.0.0/24**. Donde la dirección IP depende del alcance del router.

La configuración de los escaneos bajo la opción de Inicio Rápido es la siguiente:

- Exclusiones: **Ninguna**
- Nivel de agresividad: **Máximo**
- Denegación de Servicio: **Desabilitado**
- Fuerza Bruta: **Desabilitado**
- Conexiones Salientes de Internet: **Habilitado**
- Interfaz: **Predeterminado**
- Máxima Cantidad de Puertos: **100**
- Escaneo en Paralelo: **50**
- Ataque en Paralelo: **30**
- Acciones Posteriores: **Ninguna**
- Frecuencia: **Una vez**
- Fecha de Agendado: **Ninguna**

Evaluación

Evaluación es donde se configura un escaneo más avanzado.

General

Se parte definiendo metadata del escaneo.

Título Evaluación

Corresponde al nombre de la evaluación. Este aparece en los reportes generados , como HMTL. Este también aparecerá en el output de la API al llamar a /api/scan/status

Descripción Evaluación

Corresponde a la descripción de la evaluación. Es una descripción libre sobre lo que trata la evaluación. Similar al **Título Evaluación**, también se muestra en el output de la API al llamar a /api/scan/status

Nombre del Ingeniero

Este campo se utiliza para describir la persona encargada de correr la evaluación. Similar al **Título Evaluación**, se muestra en los reportes y también en el output de la API al llamar a /api/scan/status

Objetivos

Los objetivos corresponden a los endpoints que serán escaneados y analizados. Estos puede ser direcciones IP, Dominios o Subdominios.

NERVE no realiza descubrimiento DNS para un dominio principal determinado. Existen herramienta que potencialmente podrían utilizarse en conjunto con NERVE para lograr esto. Algunas de estas son Passive DNS, RiskIQ, Fierce, OWASP Amass, etc.

Redes

Este campo utiliza una lista de redes CIDR (separadas por coma). Por ejemplo: **212.199.1.0/24, 212.199.2.0/24**

Redes Excluidas

Este campo utiliza una lista de redes CIDR (separadas por coma). Por ejemplo: **212.199.1.1/32, 212.199.2.1/32**. Nota: Cualquier red definido aquí sera excluída de la evaluación.

Dominios

Este campo toma una lista de registros DNS (separados por coma). Por ejemplo: **ejemplo.com, sub.ejemplo.com**

Configuración

Aquí es donde se define la configuración avanzada del escaneo.

Nivel de Agresividad

Este campo decide que reglas (un ejemplo de regla es chequear que SSH este abierto) se efectúan sobre los objetivos durante un escaneo.

Por ejemplo, una regla que efectúa varias solicitudes HTTP GET solo se ejecuta en el nivel de agresividad más agresivo. Por otro lado, una regla que chequea si un puerto está abierto se ejecuta bajo cualquier condición.

Permitir Conexiones Salientes

Algunas reglas requieren conexiones salientes a internet para chequear, por ejemplo, contra listas de CVEs.

En el caso de correr internamente NERVE en una red sin internet, esta opción simplemente se puede desactivar.

Intentar Ataques de Fuerza Bruta

Algunas reglas pueden correr ataques de fuerza bruta contra ciertos servidores. Por ejemplo, si un servidor posee un puerto SSH abierto y también acepta contraseñas como medio de autenticación, entonces NERVE puede intentar autenticarse utilizando una lista de usuarios y contraseñas.

Es preciso señalar que si se provee una lista de usuarios y contraseñas vía la Opción de Diccionario, puede que la herramienta demore bastante en completar la evaluación completa. Utilice con precaución.

Intentar Ataques de Denegación de Servicio

Algunas reglas pueden correr pruebas de Denegación de Servicios contra servidores vulnerables. Es importante desabilitar esta opción para no afectar los servicios atacados.

Ethernet

Permite definir la Ethernet utilizada para escanear los puertos. Esta es opcional, si no se especifica se utilizará el valor por defecto.

Cantidad Máxima de Puertos

Esta opción es relativamente importante. NERVE empieza su evaluación ejecutando un escaneo de puertos. Mientras más puertos se escaneen, menos probable es omitir cosas.

Sin embargo, escanear muchos puertos tiene un costo asociado (tiempo), por lo tanto, se recomienda escanear solo los principales 1000-4000 puertos. Es importante encontrar el balance que funciona para cada usuario. Muy pocos puertos puede resultar en omitir varios servicios, pero el escaneo será más rápido. En tanto, muchos puertos puede significar un gran sacrificio de velocidad.

El valor por defecto para la máxima cantidad de puertos es **100** puertos (esto asegura que al menos http/https/ftp/ssh sea cubierto).

Puertos Personalizados

También se puede especificar una lista personal de puertos. NERVE empieza la evaluación ejecutando un escaneo de puertos. Mientras más puertos se escaneen, menos probable es omitir cosas.

En el caso de no especificar una lista personalizada de puertos, se utilizará por defecto una Máxima Cantidad de Puertos de **100**.

Al especificar **ambos** Máxima Cantidad de Puertos y Puertos Personalizados, **Puertos Personalizados tomará prioridad** y Máxima Cantidad de Puertos simplemente será ingnorada.

Escaneo en Paralelo

Este valor representa la cantidad de máquina que serán escaneadas simultáneamente, por ejemplo, 60 significa que el escaneo de puertos correrá contra 60 máquinas al mismo tiempo.

El valor por defecto de los Escaneos en paralelo es **50**

Ataque en Paralelo

Este valor representa la cantidad de máquina que serán atacadas simúltaneamente, por ejemplo, 60 significa que el ataque de las reglas se efectuará en 60 máquinas al mismo tiempo.

El valor por defecto de los Ataques en Paralelo es **30**

Diccionario

El diccionario permite ingresar una propia lista de usuarios y credenciales.

Cuando la opción **Intentar Fuerza Bruta** se encuentra habilitada, se utilizará la lista ingresada contra cualquier servicio identificado que permita ataques de fuerza bruta.

En el caso que no se provea una lista de usuarios y contraseñas, simplemente se utilizará una lista propia (bien corta) de usuarios y contraseñas.

Estos ataques son una especie de chequeo de sanidad, más que un ataque de fuerza bruta completo e integral. La meta no es probar millones de contraseñas, puesto que esto no es eficiente.

Al proveer una lista, asegurarse que las entradas estén separadaos por un nueva línea, p. ej.

```
password123
admin123
```

Agendar

Agendado permite determinar cuando y donde se quiere efectuar una evaluación. Existen 3 opciones: **Una vez**, **Agendar** y **Continuo**

Una vez significa que el escaneo se efectuará una vez inmediatamente, y después terminará.

Agendar significa que se puede agendar la fecha en la que se efectuará el escaneo.

Un calendario se desplegará al utilizar este modo. La fecha de ejecución del escaneo se puede seleccionar en el calendario deplegado o inrgesar directamente en el campo de fecha.

Continuo significa que un escaneo se efectuará y luego otro lo seguirá, hasta el fin de la humanidad.

Al efectuar un escaneo en el modo Continuo, un indicador "Estado" aparecerá en la barra lateral (barra del Menú) indicando que el sistema correrá por siempre.
Para detener la ejecución, simplemente hacer click en el nuevo botón que aparecerá en la barra lateral llamado **Resetear Sistema**.

Para alcanzar verdadera Seguridad Continua, se recomiendo utilizar la herramienta en el modo Continuo.

Acciones Posteriores

Se puede configurar NERVE para realizar acciones posteriores a un evento.

Web Hook

Un webhook es un evento post-procesamiento que ocurre un tiempo después de que se cumple una condición (como por ejemplo, después de terminar un escaneo).

NERVE puede enviar la información de la evaluación tan rápido como termina el escaneo a un endpoint determinado. El endpoint debe permitir el acceso de NERVE y recibir datos JSON.

Reportes

NERVE soporta 3 tipos de reportes. Estos reportes se encuentran disponibles vía la Interfaz Web de la herramienta.

- HTML
- TXT
- CSV
- XML

Todos los reportes son guardados en disco en **/opt/nerve/reports** por si se quieren revisar los reportes históricos.

En el caso de querer obtener los resultados de la evaluación vía la API, utilizar el endpoint /api/scan/status

Notificaciones

NERVE entrega notificaciones por 2 métodos: **Email** o **Slack**.

Estas opciones se encuentran en el menú superior derecho (Usuario) -> Configuraciones

Para email, se puede utilizar Amazon SES o un proveedor diferente. Se recibirá un archivo adjunto con los resultados en formato JSON una vez terminado el escaneo.

Para slack, utilizar un "incoming webhook" y pegar la URL que Slack entrega al final del proceso. De este modo, será posible recibir notificaciones de slack con los resultados una vez finalizado un escaneo.

En el caso de querer obtener los resultados de la evaluación vía la API, utilizar el endpoint /api/scan/status

Seguridad

Existen algunos mecanismos de seguridad implementados en NERVE que son importantes de considerar.

- **Content Security Policy** - Un encabezado de respuesta que controla desde donde serán cargados los recursos del escaneo.
- **Otras Políticas de Seguridad** - Estos encabezados de respuestas se encuentra habilitados: Content-Type Options, X-XSS-Protection, X-Frame-Options, Referer-Policy
- **Protección de Fuerza Bruta** - El usuario será bloquea al fallar 5 veces el inicio de sesión.
- **Protección de Cookies** - Flags de seguridad para Cookies son utilizadas, como SameSite, HttpOnly, entre otras.

En caso de identificar una vulnerabilidad, por favor notificar el bug en GitHub.

Autenticación

La autenticación a través de la API es realizada utilizando Autenticación Básica.

La Autenticación Básica es un esquema simple de autenticación en base al protocolo HTTP. El cliente envía una solicitud HTTP con el encabezado de Autorización, el cual contiene la palabra "Basic" seguido por un espacio y un string usuario:contraseña codificado en base64.

Con cada llamada a la API es necesario pasar credenciales. A continuación se muestra un ejemplo con Python y la librería de solicitudes:

```
import requests
from requests.auth import HTTPBasicAuth
username = 'admin'
password = 'admin'
requests.post('https://172.21.50.1/scan', auth=HTTPBasicAuth(username, password), json={'key':'value'})
```

API Endpoints

ID	Método	Endpoint	Autenticación	Información
1.	GET	/health	Falso	Retorna el Estado de Salud del Servidor
2.	POST	/api/scan	Verdadero	Sube una Evaluación
3.	GET	/api/scan/status	Verdadero	Retorna el Estado de la Evaluación y los Resultados
4.	PUT	/api/scan/reset	Verdadero	Resetea el Servidor. Roll Back.
5.	GET	/api/exclusion	Verdadero	Retorna el la lista de exclusiones actual
6.	POST	/api/exclusion	Verdadero	Sube la lista de exclusiones
7.	GET	/api/update/platform	Verdadero	Chequea si existen actualizaciones disponibles

C.3. Agregar nuevos *scripts*

Agregar nuevos scripts

Scripts personalizados pueden ser agregados a la herramienta por los usuarios. NERVE por defecto acepta scripts en un formato específico de Python, en la carpeta "rules" se encuentran algunos scripts en este formato, los cuales pueden ser utilizados como referencia la hora de desarrollar nuevos scripts. Para agregar este tipo de scripts a NERVE, simplemente hay que almacenar los archivos en una de las subcarpetas de "rules".

NERVE también permite la ejecución de scripts en el formato Nmap Scripting Engine(NSE). Los scripts en este formato también pueden ser añadidos a la herramienta siguiendo los pasos mencionados a continuación. Sin embargo, es importante destacar que dependiendo de la cantidad de información adicional que provean estos script es que algunas funcionalidades de la herramienta puedan ser limitadas. Adicionalmente, también es requerido modificar la herramienta para poder incorporar scripts y aprovechar todas las funcionadlidades de esta.

Pasos

1. Agregar el nuevo script a la herramienta.

Si el script se encuentra localizado en la carpeta de scripts de Nmap, entonces solo el nombre del script debe agregarse a la variable **NMAP_SCRIPTS_IN_ASSESSMENT** del archivo **config.py**.

En otros casos, agregar el script a la carpeta "nerve/rules/nse" .

2. **[Opcional]**Agregar parámetros aceptados por el script en el archivo 'config.py'. Este valor será utilizado en escaneos futuros. Cada vez que se modifique el valor la aplicación debe ser reiniciada para aplicar los cambios.

Este parámetro se le entrega al script mediante una variable definida en el archivo 'config.py', el formato utilizado para definir la variable es el siguiente:
NOMBRE_DEL_SCRIPT_NOMBRE_DEL_PARAMETRO = valor

Donde:

- 'NOMBRE_DEL_SCRIPT' es el nombre del archivo del script.
- 'NOMBRE_DEL_PARAMETRO' es el nombre del parámetro aceptado por el script.
- 'valor' es el valor del parámetro que se utiliza al correr el script.

Otras cosas a considerar:

- El nombre de la variable queda definido por la unión del 'NOMBRE_DEL_SCRIPT' con el 'NOMBRE_DEL_PARAMETRO' utilizando el carácter '_'.
- La variable puede ser definida con los caracteres en mayúscula o minúscula.
- Es necesrio borrar el tipo del archivo del 'NOMBRE_DEL_SCRIPT'. Por ejemplo, si es archivo se llama 'ftp_exploit.nse' debería dejarse como 'FTP_EXPLOIT'.
- Si el nombre incluye el carácter '-' debe ser reemplazado por el carácter '_'. Por ejemplo, si el archivo se llama 'ftp-exploit' entonces el resultado sería 'FTP_EXPLOIT'.
- Por último, si el nombre del parametro incluye el carácter '.' es necesario reemplazarlo por '_'. De este modo, el parámetro 'brute.credfile' queda como 'BRUTE_CREDFILE' al transformar.

A continuación se muestra un ejemplo abarcando todas las reglas mencionadas con anterioridad: Para **el archivo 'ftp-steal.nse'** con **parámetro 'brute.credfile'** y **valor './credfile'**. La variable resultante quedo como **'FTP_STEAL_BRUTE_CREDFILE = ./credfile'**

3. [Opcional] Agregar soporte a diferentes tipos de outputs provistos por el script. Esto permite a la herramienta determinar correctamente cuando el script encuentra una vulnerabilidad.

Esta funcionalidad puede lograrse modificando la función 'verify_output' en el archivo 'core/nse_scripts.py'. Cabe mencionar que es requerido conocimiento de programador para parsear los diferentes tipos de outputs.

4. [Opcional] Agregar metada de los scripts. Los campos soportados por NERVE son los siguientes:

- **Descripción:** Corresponde a la descripción del ataque o análisis efectuado por el script.
- **Severidad:** Indica el nivel de severidad del script. En otras palabras, que tan crítica es la vulnerabilidad asociado. Va desde 0 - 6, donde:
  - 0: Informacional
  - 1: Bajo
  - 2: Medio
  - 3: Alto
  - 4: Crítico
  - 5: Indeterminado
  - 6: Indefinido
- **Confirmación:** Detalle de la vulnerabilidad encontrada.
- **Mitigación:** Pasos necesarios para mitigar la vulnerabilidad.
- **Intesidad:** Equivalente al 'nivel de agresividad' utilizado por NERVE. Corresponde al impacto de los scripts en los objetivos de los escaneos. Su valor se encuentra en el rango 0 - 3. Donde:
  - 0: Gentil
  - 1: Medianamente Agresivo
  - 2: Agresivo
  - 3: Extremedamente Agresivo

C.4. Documentación desarrolladores

La explicación de la estructura interna de la herramienta, ya se detalla previamente en la sección Arquitectura Nerve. En esta sección también se explican los principales módulos de la herramienta y el flujo general. La mayoría de esta información es la misma que la documentación, por ello se omite en la presente sección.

En cuanto a la documentación del código, se prefirió el formato *reStructuredText*(reST) recomendado por PEP287 [30, 31]. Se aplicó este estándar solamente a las funciones desarrolladas durante este trabajo. El resto del código de la herramienta no fue documentado por razones de tiempo.