Towards Lightweight Mobile Pentesting Tools to Quickly Assess Machine Security Levels

I. Llaneza, J. Redondo, and L. Vinuesa

***Abstract*—Security audits are required to maintain appropriate security levels on computing device infrastructures. These audits require specialized tools, but sometimes they turn difficult to use when the infrastructure location, computational resources, or characteristics do not suit their requirements. This paper describes *Lightpen*, a security audit tool able to perform on-demand and fully customizable lightweight security tests from a mobile device. It enables agile security auditing tasks from any place, giving auditors full control about its activities so they adapt to the audited devices capabilities. This way, some serious security vulnerabilities may be discovered and mitigated quicker, also saving resources and time. *Lightpen* is also very modular, using reflection to incorporate features dynamically.**

***Index Terms*—Mobile device, Customization, Pentesting, Lightweight, Reflection.**

I. Introducción y Planteamiento del problema

LOS ataques a sistemas informáticos son problemas complejos y graves. El número de ataques a infraestructuras de distintos tipos [1] es mayor cada año [2], afectando a empresas privadas, públicas o infraestructuras militares [3]. Sus consecuencias son variadas: robos de información, alteraciones en los servicios [4] [5], secuestro de datos [6], toma de control de infraestructuras para fines maliciosos [7], ...

Las auditorías de seguridad (*pentesting* [8]) evalúan la capacidad de resistir ataques. En ellas profesionales contratados (*pentesters*) aplican las mismas técnicas que posibles atacantes, pero en un entorno controlado y bajo condiciones preestablecidas. Su resultado se plasma en un informe donde se describen los problemas encontrados, el nivel de exposición a ataques y posibles soluciones.

Existen herramientas muy potentes que facilitan estas auditorías, como `NMap` [9] o las *herramientas de exploración automática de vulnerabilidades*, software muy complejo con capacidades de detección muy avanzadas [10]. No obstante, su uso plantea una serie de problemas si se desean realizar auditorías rápidas. Por ejemplo, es necesario un estudio exhaustivo de sus opciones para controlar exactamente qué hacen. En ocasiones hay vulnerabilidades muy graves a las que se debe reaccionar de forma ágil, pero la herramienta no es capaz de detectarlas en un periodo de tiempo corto. Finalmente, el consumo de recursos podría impedir usarlas en algunos escenarios. `Nessus`, por ejemplo, recomienda usar 8Gb de RAM y una CPU de 4 cores a 2Ghz.

Esta falta de agilidad es un problema en pre-evaluaciones rápidas de infraestructuras que determinen si tienen vulnerabilidades muy graves. También lo puede ser en arquitecturas de *dispositivos IoT* (*Internet of Things*) [11], que podrían estar localizados en lugares poco accesibles o LANs de rango o acceso limitado, y cuyas capacidades de cómputo y conectividad son más reducidas. Esto hace necesaria una selección cuidadosa y limitada de las pruebas que pueden realizarse sobre ellos. El uso de IoT es cada vez mayor y, desgraciadamente, supone un foco de problemas graves de seguridad debido a malas configuraciones y falta de actualizaciones o de soporte por sus fabricantes [12] [13]. Por todo ello, la investigación planteada pretende resolver estos problemas mediante una herramienta (*Lightpen*) creada con los siguientes objetivos:

- Se ejecutará en un dispositivo móvil `Android` sin necesitar permisos de `root` para facilitar examinar infraestructuras en cualquier momento y lugar.
- Permitirá realizar un conjunto de test ampliable, completamente configurable y de ejecución muy automatizada para facilitar su agilidad. El auditor sabrá exactamente qué tipo de pruebas van a hacerse sin requerir un estudio complejo de parámetros, opciones y posibilidades de uso.
- El diseño de los test será versátil (operaciones a nivel de SO, red y de servicios comunes (Ej.: HTTP) y se centrará en aquellos capaces de localizar vulnerabilidades graves.
- Los tests primarán la sencillez, tendrán una ejecución rápida y un consumo de recursos moderado, adaptándose a las capacidades de cómputo y ancho de banda de dispositivos móviles.
- La información devuelta debe ser completa, descriptiva y basada en los estándares y clasificación OWASP [14] para poder ser usada en un informe de *pentesting*, o bien como guía para auditorías más detalladas.
- Debe crearse un protocolo que permita la incorporación de nuevos test de seguridad a la herramienta sin requerir su recompilación, facilitando además la distribución de nuevos test a usuarios que los necesiten.

Por tanto, la investigación planteada propone la creación de un sistema de auditorías rápidas portable, altamente configurable, ampliable y personalizable que se ejecute en sistemas de alta movilidad. El auditor tendrá en todo momento control exacto de las pruebas a realizar, con la opción de realizar sus propias variantes si lo desea. Con ello, queremos dar una mejor respuesta a situaciones que requieren respuestas rápidas y ágiles para minimizar sus consecuencias como, por ejemplo:

- Sistemas sin soporte o con problemas muy graves, para proceder a su desactivación o aislamiento inmediato.
- Versiones antiguas de software con errores de seguridad graves conocidos y documentados [15], para poder actualizarlos lo antes posible.
- Detectar errores de configuración o fugas de información.

Universidad de Oviedo, España, e-mail: uo206367@uniovi.es.
Universidad de Oviedo, España, e-mail: redondojose@uniovi.es.
Universidad de Oviedo, España, e-mail: vinuesa@uniovi.es.

- Averiguar cuando un sistema da demasiada información sobre sí mismo, algo que puede facilitar otros ataques.
- Detectar equipos o servicios desconocidos o no deseados, algo importante en caso de presencia de dispositivos *IoT*. Esto podría indicar la presencia de *malware*.

La versatilidad y utilidad de *Lightpen* depende de los test de seguridad que ejecute. Para lograr los objetivos cada test debe tener un fin concreto, delimitando sus funciones y presentándolos en forma de unidades autoexplicativas y autocontenidas. De esta manera el *pentester* sabrá claramente en qué consiste cada prueba y sus efectos. Para ello, cada prueba se presenta como un *plugin* independiente. El número de *plugins* incorporados es limitado (ver sección III-D) en una primera prueba de concepto. No obstante, la capacidad de crecimiento de la herramienta permitirá incorporar nuevos plugins de forma sencilla, y personalizarlos según las necesidades de los usuarios. Por ello, la herramienta es *open source* (cualquier usuario puede auditarla, modificar o ampliarla), y la carga de sus *plugins* se hace bajo demanda, con un módulo central que se comunica con los *plugins* mediante un protocolo definido.

Por tanto, *Lightpen* consta de 3 partes: un *cargador de plugins* que carga los *plugins* existentes bajo demanda mediante reflexión estructural [16], los *plugins* en sí y un conjunto de funcionalidades que faciliten el desarrollo de nuevos *plugins*. Todo *plugin* debe cumplir con una interfaz de servicios prefijada que establece un protocolo de comunicación entre la aplicación y cualquier *plugin* futuro. Además de flexibilidad, la reflexión estructural permite cargar solo los test que van a usarse, consumiendo menos recursos que una estructura monolítica y siendo más adecuado para plataformas móviles.

El resto del artículo se estructura así: la sección II describe herramientas relacionadas con los objetivos de nuestra investigación, destacando aquellos aspectos en los que no los cubren. La sección III describe la arquitectura de la aplicación diseñada para validar los objetivos de investigación. La sección IV explica la ejecución del prototipo actual sobre objetivos reales, cómo ha ayudado a la detección rápida de problemas de seguridad y su consumo de recursos. Finalmente, la sección V detalla las conclusiones y el trabajo futuro.

II. Trabajo relacionado

A. Herramientas de Detección Avanzadas

Existe una amplia literatura e investigación sobre herramientas capaces de detectar incidencias en infraestructuras de máquinas [17]. Las aproximaciones que siguen estas herramientas varían: algunas usan *hardware* específico que hace parte del procesamiento, agentes (todos los servidores monitorizan y se coordinan para responder un ataque), lenguajes específicos de dominio (DSLs) para el desarrollo de sistemas de intrusión, o diversas técnicas de inteligencia artificial, entre las que se destacan los sistemas de reglas. Sistemas como *Bro*, *EMERALD*, *Janus* o *CMS*, entre otros, son algunos ejemplos [17]. Si bien cumplen con su objetivo de forma muy efectiva, su arquitectura y propósito hace que necesiten un despliegue complejo y unas capacidades de cómputo elevadas para poder desempeñar su función, lo que las hace inadecuadas para ser usadas en el escenario planteado. Por otro lado, su uso no fomenta la intervención directa y puntual del auditor para hacer pruebas específicas, sino que están pensadas para estar presentes de forma constante en una infraestructura, supervisarla y emitir alertas o realizar acciones en caso de encontrar anomalías.

Existen herramientas de detección automática de problemas de seguridad de carácter más activo, permitiendo al *pentester* obtener información bajo demanda. Sin embargo, realizan estudios exhaustivos a costa de requerir muchos recursos computacionales y tiempo, sin que tampoco permitan un control fino de los test ejecutados. Esto las hace inadecuadas para ejecutar test concretos y ágiles desde plataformas móviles. Ejemplos son *OpenVAS* [18], *Nessus* [19], *Nexpose* [20], *OWASP ZAP* [21] o *w3af* [22]. El concepto de *plugins* planteado en *Lightpen* está basado en la forma que estas dos últimas permiten ampliar sus funcionalidades.

B. Herramientas Ligeras para SO Escritorio

Existen una serie de herramientas que, si bien tradicionalmente se ejecutan en SO de escritorio, podrían hacerlo en dispositivos móviles / IoT gracias a su consumo de recursos relativamente bajo. Algunas de ellas están especializadas en el análisis de servidores web, como *Nikto* [23]. Sus principales inconvenientes son la elevada especialización de los test que realizan y la imposibilidad de personalizar los mismos. Existen a su vez herramientas mucho más generalistas, como la popular *NMap* [9]. Sin embargo, se requiere un estudio en profundidad de su gran cantidad de opciones para poder personalizar el análisis de seguridad a ejecutar, restándole agilidad al proceso. Además, su GUI oficial carece de opciones que permitan explotar toda su funcionalidad, lo que fomenta la aparición de GUIs no oficiales [24]. Por tanto, ambos tipos de herramientas plantean inconvenientes a la hora de realizar test personalizados de forma ágil y sencilla.

C. Herramientas para Dispositivos Móviles

Actualmente es posible encontrar herramientas de auditoría para dispositivos `Android` en su *market* oficial *Google Play*, y que cuentan por tanto con una elevada movilidad. Algunas de ellas, como `Droidbug Pentesting`, son distribuciones de *Linux* sobre `Android`, con múltiples herramientas de *pentesting* populares que deben usarse individualmente. Este tipo de aproximaciones no cumplen con los requisitos de agilidad y automatización planteados, al obligar al usuario a conocer el funcionamiento y parámetros de cada herramienta por separado. Si bien su carácter de SO permite la instalación de nuevas herramientas y opciones, también hace que el consumo de recursos sea mayor.

Otras herramientas están enfocadas solamente al *pentesting* web, como `Kayra the Pentester Lite`, un escáner de vulnerabilidades para servidores HTTP que incluye variedad de tests y permite enviar los resultados vía e-mail. Además, podemos mencionar `HTTP Tools`, un conjunto de herramientas que permiten el *footprinting* de servidores web remotos, realizar algunos test concretos de seguridad y la obtención de información acerca de las peticiones HTTP que

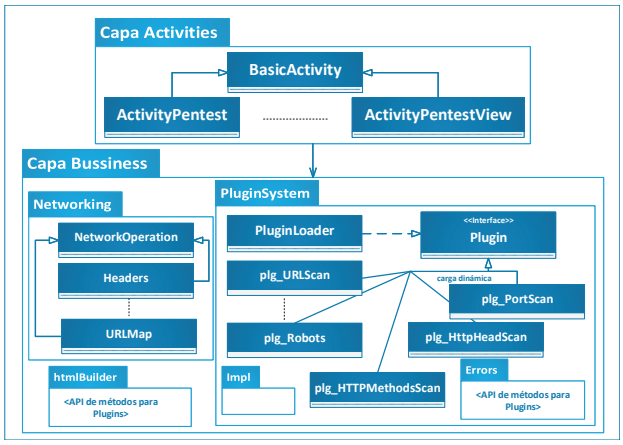

Fig. 1. Arquitectura de *Lightpen*.

aceptan y sus webs alojadas. Si bien estas herramientas permiten la realización de análisis concretos y ágiles, su elevada especialización (sólo servidores HTTP) y falta de capacidades de ampliación hacen que no cumplan con los objetivos de versatilidad y personalización especificados.

Una tercera categoría de herramientas está formada por aquellas que están especializadas en la localización de información de sistemas remotos, integrando servicios como `whois`, `traceroute` y extracción información del servidor DNS acerca de un objetivo, entre otros. Un ejemplo de este tipo de herramientas es `Hackode`. Si bien cubren adecuadamente una serie de necesidades específicas, como localizar la presencia de equipos en localizaciones donde no deberían estar activos, de nuevo su falta de versatilidad supone un problema para cubrir los objetivos planteados.

Finalmente, podemos encontrar aplicaciones con funcionalidades para explotar vulnerabilidades de los sistemas que analizan. Ejemplos de este tipo de herramientas son `Info P. D. S.` o `Andro Hackbar`, especializada en la inyección de ataques *XSS* o *SQL Injection*. Estas son herramientas de naturaleza distinta a *Lightpen*, puesto que nuestros objetivos de diseño son fundamentalmente la localización y prevención ágil de vulnerabilidades, no su explotación.

III. Solución propuesta

La arquitectura general de *Lightpen* consta de varios módulos principales representados en la Figura 1.

A. Cargador de Plugins

La parte central de la aplicación es el cargador dinámico de los *plugins* que modelan cada test de seguridad (`PluginLoader`), ocupándose también de gestionarlos y orquestarlos. Además, recoge los resultados de su ejecución para suministrárselos al módulo encargado de presentar la información (`htmlBuilder`). Los *plugins* seleccionados por el usuario serán por tanto los que tengan la mayor parte de la carga computacional en cada auditoría. El consumo de recursos se optimiza así al cargarse y ejecutarse sólo los *plugins* que el usuario desea, cumpliendo con los objetivos indicados en la sección I.

El cargador de plugins hace a *Lightpen* altamente modular, aislando la gestión de operaciones de seguridad de su ejecución. Esto hace que *Lightpen* sea una aplicación adaptativa en tiempo de ejecución, facilitándose su evolución, personalización y mantenimiento. La construcción de nuevos *plugins* permite incorporar nuevas funcionalidades y su distribución a otras instalaciones de *Lightpen*. Los errores corregidos o mejoras en las operaciones de seguridad están contenidos en el código de cada *plugin*. Ninguna de estas operaciones requiere cambiar ni recompilar el código de la aplicación principal ni de otros *plugins*.

Para lograr este objetivo se usan los mecanismos de introspección [16] del lenguaje *Java*, a través de las funcionalidades del paquete `java.lang.reflect`. La carga de nuevos *plugins* es dinámica gracias al método `Class.forName(nombreDeUnaClase:String)`, que cargará las clases cuyo nombre le indiquemos como parámetro. *Lightpen* requiere que todo *plugin* susceptible de ser cargado dinámicamente esté en una ruta configurable.

B. Estructura de los Plugin y Normas para su Creación

Como se indicó en la sección I, cada prueba de seguridad estará contenida en su *plugin* (ver subsección III-D), configurando su ejecución según las preferencias de usuario suministradas por el cargador. *Lightpen* facilita la creación de nuevos *plugins* siguiendo una serie de normas de programación. Toda clase que las cumpla y que se encuentre en la ruta mencionada anteriormente se podrá cargar como *plugin*. Esto facilita la creación de nuevos test y su distribución a otras instalaciones simplemente copiando sus ficheros.

Los *plugins* se estructuran en dos partes. La primera es una clase encargada de suministrar información e interactuar con la aplicación, implementando los servicios de la interfaz `Plugin` (ver Figura 2). El nombre de esta clase debe comenzar por `plg_` para distinguirla de otras clases auxiliares. Esta clase es cargada dinámicamente por `PluginLoader` para informar de la presencia del plugin y su funcionalidad, pero no carga su implementación. Esta reside en otra clase situada en el paquete `impl` (Figura 2) y se carga dinámicamente al llamar a `run`, comunicándose con ella a través de la interfaz `IPluginImplementation`. De esta forma, la interacción (`plg_<NewPlugin>`) y la implementación (`plg_<NewPlugin>Impl`) de cada test de seguridad están separadas. Esta última sólo se carga si realmente va a usarse, disminuyendo así el consumo de recursos y cumpliendo nuestros objetivos. La implementación puede usar opcionalmente los paquetes `networking`, `errors` y `htmlBuilder` que son APIs de servicios comunes reutilizables por los *plugins*, descritos en la siguiente subsección.

C. API de Operaciones Comunes para el Desarrollo de Plugins

Aunque cada *plugin* implementa un test de seguridad distinto, existen una serie de operaciones auxiliares que muchos *plugins* pueden necesitar. Estas operaciones comunes se han identificado y se ofrecen en forma de API para ahorrar trabajo a los implementadores de *plugins*. Así se

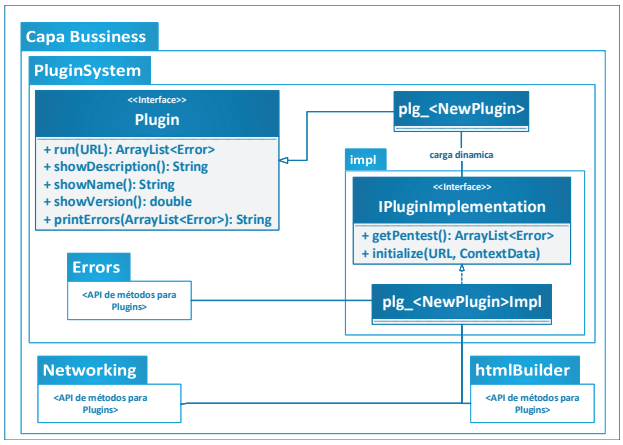

Fig. 2. Estructura de un plugin y los servicios a implementar.

permite una base de desarrollo común facilitando su depuración y optimización. Esta API contiene operaciones de red (paquete `networking`), funcionalidades de gestión de errores (paquete `errors`) y para mostrar los resultados de las operaciones (paquete `htmlBuilder`). A modo de ejemplo, la Tabla I describe las operaciones auxiliares principales del paquete `networking` y su función. Como se muestra en la Figura 1, todas ellas derivan de la clase `NetworkOperation`.

TABLA I
OPERACIONES PRINCIPALES DEL API PARA EL DESARROLLO DE PLUGINS

Operación	Descripción
`Headers`	Obtiene las cabeceras de una petición HTTP
`Resources`	Dada una conexión, obtiene el código HTML de una web
`HTTP Methods`	Obtiene los métodos HTTP soportados por un servidor remoto
`URLMap`	Permite navegar por enlaces encadenados, con un nivel de profundidad configurable. Además, mediante el uso de un patrón de diseño `CommandExecutor` permite ejecutar diversas acciones sobre los mismos, que también pueden ser creadas por el desarrollador
`Ports`	Escáner de servicios TCP y UDP

D. Plugins de Seguridad

El número de *plugins* que pueda usar *Lightpen* en un momento dado determinará la utilidad de la aplicación en diferentes escenarios. Es necesario por tanto hacer una selección de *plugins* inicial cuidadosa, para garantizar que la aplicación cubrirá unos mínimos de cara a resultar útil en el mayor número de situaciones posible. Como se mencionó en la sección I, los *plugins* se han diseñado con la intención de que las operaciones que realicen sean autocontenidas, necesiten pocos recursos, estén enfocadas a la localización de errores graves y terminen su ejecución en poco tiempo, adaptándose así a las posibilidades de un dispositivo móvil. La siguiente lista muestra los *plugins* incluidos en el prototipo de *Lightpen*.

1) `HTTP Headers`: Analiza las cabeceras de respuesta HTTP para determinar si se han activado determinados aspectos de seguridad (uso de `HSTS`, `HKPK`, habilitación de protecciones contra `XSS...`).
2) `Robots.txt`: Determina si existe un fichero `robots.txt` y analiza su contenido por si contuviese información que facilitase ataques.
3) `URL Mapper`: Crea un mapa de recursos asociados a una URL y permite visualizarlos para comprobar que no se están enlazando URLs no deseadas.
4) `HTTP Methods`: Analiza los métodos HTTP soportados por un servidor, para localizar aquellos que puedan suponer un problema de seguridad.
5) `Ports`: Analiza puertos TCP y UDP en busca de servicios en ejecución que puedan comprometer la seguridad. Se usa una lista de puertos mundialmente más atacados, incluyendo aquellos que lo son más del 0,005% de las veces. Además, reconoce el nombre del servicio más comúnmente asociado a los mismos.
6) `Error Page`: Comprueba la existencia de páginas de error por defecto que puedan indicar el tipo o versión de algún producto.
7) `Dir Listing`: Comprueba si el servidor tiene habilitado el listado de directorios cuando se accede a una ruta del mismo sin especificar un fichero concreto, dando demasiada información sobre sus contenidos.
8) `Comments`: Analiza los comentarios presentes en webs o ficheros, para buscar palabras clave o comentarios por defecto que puedan delatar el uso de determinados productos de terceros y/o versiones de los mismos.
9) `Device Locator`: Escaneo e identificación de dispositivos en la misma red que el objetivo, usando la caché ARP de la red.

Por otro lado, hemos proyectado la incorporación de los siguientes *plugins* en las siguientes versiones de la aplicación:

1) `HTTP Products`: Complementa `HTTP Header` analizando las cabeceras HTTP para buscar nombres y versiones de software usado en el mismo. Incluye además una búsqueda automatizada de `CVEs` [15] que describan las vulnerabilidades de esos productos, para hacer un perfil de vulnerabilidades del objetivo.
2) `Blacklist`: Comprueba si la IP o URLs asociadas a la máquina remota figura en listas negras (como *Google Safe Browsing* [25]), lo cual podría ser indicativo de que la máquina es o ha sido origen de actividades maliciosas. La funcionalidad será similar a la implementada por el *script* NSE `http-google-malware` de `NMap` [9].
3) `Phising`: Determina si existen nombres de dominio similares al del sistema remoto que puedan ser usados para intentos de *phishing*, aprovechándose de errores a la hora de teclearlo por parte de los usuarios o mediante e-mails fraudulentos. Si además este dominio es identificado como malicioso por el *plugin* anterior, el intento de *phising* sería mucho más probable.
4) `File Tips`: Comprueba la existencia de determinados ficheros o rutas conocidos, que pueden indicar el uso de determinados productos o *frameworks* de los que se sabe que usan una estructura de directorios o ficheros conocida. Para ello, se hará una selección de *frameworks*

de desarrollo web comunes (Wordpress, Joomla,...) y se hará una implementación de la técnica *static files* usada por herramientas como BlindElephant [26].

5) Metadata: Analiza la presencia de metadatos que puedan dar demasiada información en los ficheros o imágenes servidos por el sistema remoto.

Todos los *plugins* contenidos y planificados en *Lightpen* están asociados a alguna posible vulnerabilidad clasificada por OWASP para que su funcionalidad sea más clara y distinguible para sus usuarios. La Tabla II muestra esta asociación.

TABLA II
VULNERABILIDADES CUBIERTAS POR *Lightpen*

Plugin	Vulnerabilidad(es) OWASP
HTTP Header	*Test HSTS* (OTG-CONFIG-007) *Testing for reflected XSS* (OTG-INPVAL-001) *Testing for stored XSS* (OTG-INPVAL-002)
Robots.txt	*Review server metafiles for inf. leakage* (OTG-INFO-003)
HTTP Url	*Map ex. paths through app.* (OTG-INFO-007) *Enumerate server app.* (OTG-INFO-004) *Identify app. entry points* (OTG-INFO-006) *Map app. architecture* (OTG-INFO-010)
HTTP Method	*Test HTTP methods* (OTG-CONFIG-006)
Ports	*Enumerate server app.* (OTG-INFO-004) *Identify app. entry points* (OTG-INFO-006)
HTTP Products	*Enumerate server app.* (OTG-INFO-004) *Fingerprint app. framework* (OTG-INFO-008) *Fingerprint web app.* (OTG-INFO-009)
Blacklist, Phising, Device Locator	*Conduct search engine discovery / reconnaissance for inf. leakage* (OTG-INFO-001)
File Tips	*Test file extensions handling for sensitive information* (OTG-CONFIG-003) *Review old, backup and unreferenced files for sensitive information* (OTG-CONFIG-004)
Error Page	*Testing for error code* (OTG-ERR-001)
Dir Listing	*Test file Extensions handling for sensitive information* (OTG-CONFIG-003) *Review old, backup and unreferenced files for sensitive information* (OTG-CONFIG-004)
Comments, Metadata	*Review webpage comments and metadata for information leakage* (OTG-INFO-005)

E. Evaluación de Resultados

Lightpen se ha evaluado siguiendo una metodología cualitativa a la hora de comprobar la efectividad de los *plugins* implementados por el prototipo actual. Para ello, se contactó con una serie de empresas/instituciones públicas que nos autorizaron a realizar pruebas sobre sus servidores web, de manera que pudiésemos comprobar si todos los *plugins* consiguen los objetivos prefijados. Varias empresas nos autorizaron a ello a cambio confidencialidad y proporcionarles los resultados y asesoría sobre soluciones a los problemas detectados. Entre ellas, destacamos la *Escuela de Ingeniería Informática* de la *Universidad de Oviedo* [27] (conectándose a su red Wifi) y *Mecalux Software Solutions* [28] (a través de Internet).

A la hora de calcular el consumo de recursos (CPU, memoria y tiempos de realización de operaciones) hemos seguido una metodología que permite el cálculo de manera estadísticamente correcta y que ya hemos empleado en trabajos anteriores [29]. Teniendo en cuenta que la aplicación está diseñada para consumir solamente los recursos estrictamente necesarios, hemos considerado que la mejor forma de medir su consumo es fijar un escenario de uso típico sobre el que hacer las pruebas. La aplicación fue instrumentada con una tarea en segundo plano que usa los servicios del SO para medir sus valores de % de uso de la CPU y memoria consumida. Posteriormente, hemos ejecutado la aplicación múltiples veces contra un mismo sistema remoto ejecutando el mismo conjunto de test de seguridad (compuesto por los *plugins* HTTP Header, Robots.txt y HTTP Method), habiéndose usado (y por tanto cargado) previamente dos *plugins* más (HTTP Url y Ports). Los valores de consumo se tomaron cada 100ms y, una vez la aplicación alcanzó su *steady state*, se computaron los valores medios de consumo de cada recurso mediante la media geométrica.

Por otro lado, para medir el tiempo que tarda cada *plugin* en hacer individualmente su escaneo de seguridad correspondiente se ha ejecutado cada uno contra otra máquina en la misma red local que *Lightpen*, aplicando la metodología anterior para la recogida de valores. Esto se ha hecho así en este caso para minimizar al máximo la latencia de la red y tratar de medir sólo los tiempos de cada operación.

IV. RESULTADOS

Actualmente hemos implementado un prototipo funcional de *Lightpen* para Android Marshmallow (6.0, API Level 23) o superior, por lo que puede usarse en más de un 70% de los dispositivos Android actuales. El primer prototipo ocupa 2.8Mb e implementa los *plugins* mencionados en la sección III. Requiere el permiso INTERNET para efectuar los test sobre sistemas remotos, y luego los relativos al almacenamiento para mostrar resultados de test anteriores y guardarlos (WRITE_EXTERNAL_STORAGE y READ_EXTERNAL_STORAGE). Para ejecutar un escaneo de vulnerabilidades con este prototipo, desde la pantalla principal (Figura 3) el usuario puede acceder a la funcionalidad de escaneo o ver el resultado de escaneos anteriores. Si se selecciona hacer un escaneo, *Lightpen* mostrará los *plugins* disponibles en ese momento en la instalación local, permitiendo al usuario seleccionarlos. Además, deberá indicar la dirección del sistema remoto a analizar, con la opción de analizar si se encuentra accesible antes de la ejecución. Una vez hecho esto, el sistema pasará a cargar y ejecutar los *plugins* seleccionados.

Al terminar el escaneo el usuario podrá ver el resultado del mismo. El prototipo actual permite mostrar los resultados en formato HTML estándar. La Figura 4 muestra el resultado de un escaneo hecho sobre un sistema remoto real seleccionando dos *plugins*. El riesgo asociado a cada vulnerabilidad descubierta y el color que indica su importancia sigue los estándares OWASP [30]. Se permite enviar o compartir esos resultados con otras aplicaciones que acepten el tipo MIME text/html, usando los mecanismos estándar de Android para ello.

Para la evaluación de *Lightpen* se ha seguido la metodología enunciada en la sección III-E. La Figura 4

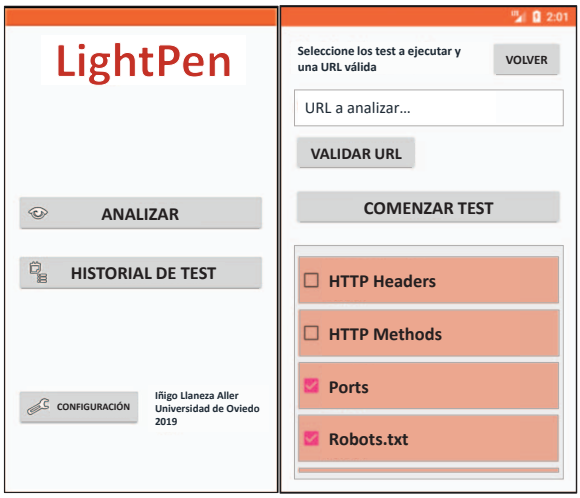

Fig. 3. Proceso para lanzar un escaneo en *Lightpen*.

muestra uno de los resultados de las maquinas reales escaneadas, anonimizado por confidencialidad. En [31] (directorio `pentestingResults`) pueden además consultarse otros informes de resultados de escaneos anonimizados, a los que haremos referencia. El servidor analizado en la Figura 4 carece de una serie de cabeceras HTTP recomendadas para mejorar la seguridad de todas sus webs, por lo que su nivel de seguridad es mejorable. Por ejemplo, la falta de `Strict-Transport-Security` no fuerza la navegación vía HTTPS, lo que posibilita la intercepción de la información introducida por sus usuarios al permitir el servidor tráfico cifrado y no cifrado. Otra vulnerabilidad potencial podría darse en caso de que se introduzca un vector de *XSS* (ataque contra el que tampoco se activa la cabecera `X-XSS-Protection`): la falta de control sobre quién puede mostrar *Frames* en la web, al no tener la cabecera `X-Frame-Options` configurada, facilita ataques de *clickjacking* y robos de información. El fichero `LightpenReportHeaders.html` muestra otro ejemplo donde ninguna de las cabeceras HTTP recomendadas se ha activado. En el análisis representado en la Figura 4 el *plugin* `Robots.txt` encontró además un fichero `sitemap.xml` cuyo contenido estaba mal diseñado y mostraba rutas correspondientes a una versión anterior de la web que ya no estaban disponibles. Si bien en este caso concreto no contenían información relevante, esto puede llevar a localizar por ejemplo ficheros de *backup* del código de la aplicación dejados por error. En ellos es posible encontrar cadenas de conexión, nombres de usuarios y contraseñas de prueba aún activos y otra información muy peligrosa. Ambos hechos fueron reportados inmediatamente tras el escaneo para que se corrigiesen.

Otra de las webs analizadas tenía una serie de problemas de configuración que revelaban, mediante cabeceras HTTP no estándar (*plugin* `HTTP Header`), versiones de servicios y sistemas operativos concretas. Tras una consulta manual a [15] con dicha información, se comprobó que esas versiones tenían vulnerabilidades graves, por lo que tuvo que dar un aviso inmediato para que se procediese a actualizarlos. Otro ejemplo que muestra la utilidad de tener una herramienta ágil ocurrió cuando a través del *plugin* `Robots.txt` encontramos un listado de recursos `.php` que nada tenían que ver con la web subidos por error (ver fichero `LightpenReportRobots.html`). Estos contenían ejemplos de tutoriales (no pensados para ser código en producción seguro) y datos que revelaban la versión de PHP usada (que vuelve a llevarnos al caso anterior si la versión tiene problemas de seguridad graves), que hubo que retirar de inmediato.

En otro de los análisis realizados, varias máquinas de una red presentaban problemas. Además de errores de configuración que revelaban demasiada información (ver fichero `LightpenReportDirListing.html`), se detectó que una máquina había sido configurada por personal sin conocimientos técnicos, con el resultado de que se estaban soportando métodos HTTP inadecuados (*plugin* `HTTP Method`) para la función que cumplía (ver fichero `LightpenReportMethods.html`). Hubo pues que proceder a desactivar de inmediato el soporte de los mismos por el riesgo que suponían. Otra de las máquinas presentaba puertos anómalos abiertos (*plugin* `Ports`), ya que solamente debería tener abiertos el 22, el 80 y el 443 al ser un servidor web (ver fichero `LightpenReportPorts.html`). Un examen manual posterior reveló que había sido comprometida a través de una vulnerabilidad SSH y tenía instalado *malware* que generaba mucho tráfico y trataba de expandirse al resto de máquinas de la red, disminuyendo la calidad de las conexiones y habiendo afectado en el momento de su detección a una máquina adicional. En estos casos, un tiempo de respuesta rápido es importante para minimizar el impacto de un ataque y *Lightpen* demostró ser una herramienta adecuada para ello.

Finalmente, destacar que en uno de estos casos se realizó una auditoría en profundidad posterior, donde la información obtenida por *Lightpen* fue utilizada para orientar a los auditores a la hora de preparar ataques (ahorrando así tiempo) y permitió localizar un problema adicional de escalado de directorios gracias a la información obtenida por el *plugin* `HTTP Url` (ver fichero `LightpenReportURLScan.html`).

Con estos resultados hemos podido constatar por tanto que los objetivos iniciales de *Lightpen* (ver sección I) se han cumplido: los *plugins* actuales proporcionan información rápida de problemas de seguridad graves para poder corregirlos rápidamente, y sirven de guía para auditorías más en profundidad que se hagan posteriormente.

Lightpen se ha ejecutado con éxito sobre *smartphones* de varias gamas con años en el mercado gracias su uso contenido de recursos: *Nexus 5* (2013, `Snapdragon 800`, 4 núcleos, 2.3Ghz, 2Gb), *Galaxy S3* (2012, `Exynos 4412`, 4 núcleos, 1.4Ghz, 1Gb), *Xperia Z* (2013, `APQ8064`, 4 núcleos, 1.5Ghz, 2Gb) y *Xperia Z5* (2015, `Snapdragon 810`, 8 núcleos, 2Ghz, 3Gb). Siguiendo la metodología de la sección III-E sobre un *Nexus 5x* (API 26), *Lightpen* tiene un consumo medio de 158Mb de RAM y la carga media de la CPU es de un 14,38%. Se permite configurar el nº de threads y la profundidad de recorrido de un sitio web para adaptarse mejor al hardware.

Finalmente, además de los tiempos (`T`) de ejecución de cada *plugin* en el escenario y con la metodología descritos

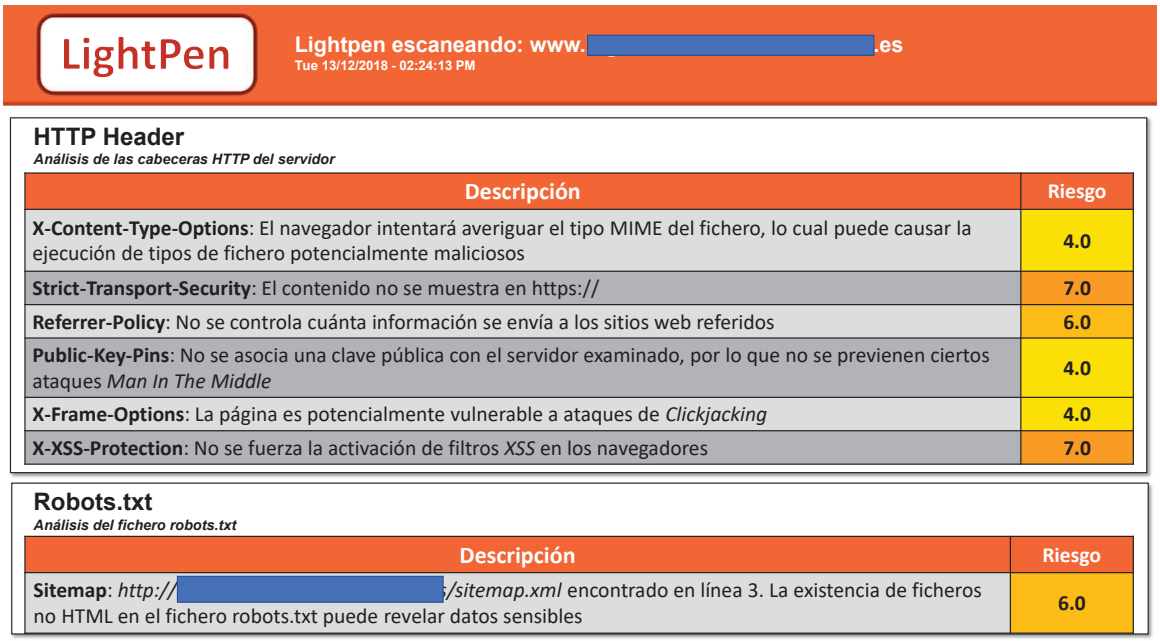

Fig. 4. Resultados de un escaneo en *Lightpen*.

TABLA III
DATOS EMPÍRICOS SOBRE LOS PLUGINS DE LIGHTPEN

Plugin	T (ms)	NCLOC	av(G)	.class
`HTTP Header`	106,98	122	1,58	1,9Kb
`Robots.txt`	102,36	130	2,5	2,6Kb
`HTTP Method`	146,38	127	2	2,3Kb
`Ports`	15093,8	44	1,14	0,7Kb
`Error Page`	124,07	114	2,46	2Kb
`Dir Listing`	130,38	81	1,8	1,4Kb
`Device Locator`	918,08	70	1,4	0,9Kb
`Comments`	135,36	100	2,18	1,8Kb

en la sección III-E, la tabla III recoge datos para comprobar que tienen una ejecución rápida y baja complejidad de diseño gracias a la librería de servicios reutilizables (ver sección III-C). Estos son las métricas `NCLOC` (líneas de código fuente, sin comentarios), `av(g)` (complejidad ciclomática media de los métodos) y el tamaño de los `.class` de la implementación de cada *plugin*. Todos los valores medidos se mantienen en rangos razonables acordes con los objetivos de nuestra investigación. El único valor que parece discordante es el del *plugin* `Ports`, que tarda 15s. Sin embargo, debe tenerse en cuenta que un análisis de puertos es en sí una operación costosa, aun con latencias de red bajas, por el tiempo que puede tardar el sistema objetivo en devolvernos una conexión (o no, si el puerto está cerrado). Precisamente para dejar los tiempos en rangos que consideramos aceptables hemos limitado el nº de puertos escaneados de la forma mencionada en la sección III-D, escaneando en total 205 puertos TCP/UDP a una media de 73ms/puerto.

V. CONCLUSIONES

Lightpen es una aplicación móvil y altamente modular que permite hacer rápidamente test de seguridad personalizados con un bajo consumo de recursos sobre equipos remotos en cualquier momento y lugar. El conjunto de test que realiza es completamente configurable, y la información que obtiene sigue estándares conocidos, además de poder ser enviada a otras aplicaciones para su procesamiento. *Lightpen* permite a los *pentesters* obtener información preliminar acerca del nivel de seguridad de un equipo remoto bajo demanda, con un control detallado de las tareas que se realizan y desde su propio dispositivo móvil. Esta información puede resultar muy valiosa para dirigir futuras operaciones de *pentesting* en profundidad, o para identificar rápidamente vulnerabilidades graves y acciones correctivas de inmediato. Por otro lado, está también especialmente pensada para escanear la seguridad de dispositivos IoT situados en localizaciones que hacen que su análisis por otros medios sea más difícil. *Lightpen* ha sido probada con éxito desde diferentes *smartphones* `Android` contra objetivos reales, alcanzando los objetivos planteados.

El trabajo futuro de esta línea de investigación explorará las posibilidades de adaptar las técnicas empleadas por las herramientas existentes de detección de incidencias mencionadas en la sección II al mismo escenario en el que se enfoca el uso de *Lightpen*. De esta forma, queremos comprobar la viabilidad de un sistema basado en agentes [17] que, mediante el uso de sistemas de reglas, intervención directa de los auditores y otras técnicas, pueda establecer labores de vigilancia de una red IoT o similar y la desactivación automática de dispositivos en los que se detecten riesgos de seguridad evidentes. En paralelo, además de incorporar todos los *plugins* de la sección III, se contempla integrar *Lightpen* con otras líneas desarrolladas por nuestro grupo de investigación [32] que puedan beneficiarse de la capacidad de descubrir vulnerabilidades de forma ágil y rápida. Esto incluye la investigación que realizamos en mejorar la aplicación de herramientas en asignaturas de administración [33] y seguridad [34] y auditar periódicamente webs en

producción desarrolladas por nuestro grupo [35]. Finalmente, también queremos lograr autorización para usar la herramienta sobre más objetivos reales. La versión actual de *Lightpen* y su conjunto de *plugins* más actual puede descargarse de [31].

AGRADECIMIENTOS

Este trabajo ha sido financiado parcialmente por el Ministerio de Ciencia, Innovación y Universidades de España, Proyecto RTI2018-099235-B-I00, y también por la Universidad de Oviedo (proyecto GR-2011-0040)

REFERENCIAS

[1] D. Harley, L. Myers, S. Cobb, and C. Gutierrez, "Cybersecurity trends 2019: Privacy and intrusion in the global village," ESET, Tech. Rep., 2018, (Dec 10, 2018).
[2] A. Bendovschi, "Cyber-attacks – trends, patterns and security countermeasures," *Procedia Economics and Finance*, vol. 28, pp. 24 – 31, 2015, 7th INTERNATIONAL CONFERENCE ON FINANCIAL CRIMINOLOGY 2015, 7th ICFC 2015, 13-14 April 2015,Wadham College, Oxford University, United Kingdom.
[3] P. M. Vidhya, "Cyber security: Threats and challenges," *Int.l J. of Computer Science and Mobile Computing*, vol. 3, pp. 586–590, 02 2014.
[4] T. Matthews, "What DDoS attacks really cost businesses," Imperva Incapsula, Tech. Rep., 2016, (Dec 10, 2018).
[5] N. A. S. Lima and M. P. Fernandez, "Towards an efficient DDoS detection scheme for software-defined networks," *IEEE Latin America Transactions*, vol. 16, no. 8, pp. 2296–2301, Aug 2018.
[6] S. Hsiao and D. Kao, "The static analysis of WannaCry ransomware," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, Feb 2018, pp. 1–1.
[7] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking," in *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, April 2018, pp. 58–66.
[8] OWASP, "Penetration testing methodologies," www.owasp.org/index.php/Penetration_testing_methodologies, 2016, (Dec 10, 2018).
[9] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Project, 2009.
[10] S. Kumar Yadav, D. Shankar Pandey, and S. Lade, "A comparative analysis of detecting vulnerability in network systems," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 7, pp. 336–340, 05 2017.
[11] A. Cravero, "Arquitecturas de big data y el internet de las cosas: Un mapeo sistemático de estudios," *IEEE Latin America Transactions*, vol. 16, no. 4, pp. 1219–1226, Apr 2018.
[12] K. Zhao and L. Ge, "A survey on the internet of things security," in *2013 Ninth International Conference on Computational Intelligence and Security*, Dec 2013, pp. 663–667.
[13] M. Abomhara and G. Køien, "Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks," *Journal of Cyber Security*, vol. 4, pp. 65–88, 05 2015.
[14] OWASP, *OWASP Pentesting Guide v4*. OWASP, 2017, vol. 1.
[15] Mitre.org, "CVE details: The ultimate security vulnerability datasource," https://www.cvedetails.com/, 2018, (Dec 10, 2018).
[16] F. Ortin, J. M. Redondo, and J. B. G. Perez-Schofield, "Efficient virtual machine support of runtime structural reflection," *Science of Computer Programming*, vol. 74, no. 10, pp. 836 – 860, 2009.
[17] S. Singh and S. Silakari, "A survey of cyber attack detection systems," *Int.l Journal of Computer Science and Network Security*, 05 2009.
[18] Openvas.org, "OpenVAS: Open vulnerability assessment system," www.openvas.org/, 2018, (Dec 10, 2018).
[19] B. Martin and C. Fennelly, "Web application scanning with nessus," Tenable Network Security, Tech. Rep., 2013, (Dec 10, 2018).
[20] Rapid7, "Nexpose: Your on-prem vulnerability scanner," https://www.rapid7.com/products/nexpose/, 2018, (Dec 10, 2018).
[21] GitHub.com, "ZAProxy DevExtending," github.com/zaproxy/zaproxy/wiki/DevExtending, 2015, (Dec 10, 2018).
[22] w3af.org, "w3af – open source web application security scanner," http://w3af.org/, 2018, (Dec 10, 2018).
[23] Cirt.Net, "Nikto," cirt.net/Nikto2, 2018, (Dec 10, 2018).
[24] J. M. Redondo and D. Cuesta, "Towards improving productivity in nmap security audits," *Journal of Web Engineering*, vol. 18, no. 7, pp. 1–38, Sep 2019.
[25] P. K. Sandhu and S. Singla, "Google safe browsing - web security," *International Journal of Computer Science Engineering & Technology*, vol. 5, no. 7, pp. 283–287, 2015.
[26] P. Thomas, "BlindElephant web application fingerprinter," http://blindelephant.sourceforge.net/, 2018, (Dec 10, 2018).
[27] E. de Ingeniería Informática, "Web oficial de la Escuela de Ingeniería Informática de la Universidad de Oviedo," ingenieriainformatica.uniovi.es/, 2018, (Dec 10, 2018).
[28] Mecalux, "Página oficial de Mecalux Esmena," https://www.mecalux.es/, 2018, (Dec 10, 2018).
[29] J. M. Redondo and F. Ortin, "A comprehensive evaluation of common python implementations," *IEEE Software*, vol. 32, no. 4, pp. 76–84, July-Aug. 2015.
[30] OWASP, "OWASP risk rating methodology," www.owasp.org/index.php/OWASP_Risk_Rating_Methodology, 2016, (Dec 10, 2018).
[31] I. Llaneza, J. Redondo, and L. Vinuesa, "Lightpen github repository," https://github.com/jose-r-lopez/LightPen/, 2019, (March 12, 2019).
[32] U. of Oviedo, "Computational reflection research group," http://www.reflection.uniovi.es/, 2019, (May 15, 2019).
[33] J. M. Redondo, "Improving student assessment of a server administration course promoting flexibility and competitiveness," *IEEE Transactions on Education*, pp. 1–8, 2018.
[34] J. M. Redondo and L. Varela, "Filesync and Era Literaria: Realistic open source webs to develop web security skills," *Journal of Web Engineering*, vol. 17, no. 5, pp. 1–22, Aug 2018.
[35] J. M. Redondo and F. Ortin, "A SaaS framework for credit risk analysis services," *IEEE Latin America Transactions*, vol. 15, no. 3, pp. 474–481, March 2017.

Íñigo Llaneza Graduado en Ingeniería Informática del Software por la Universidad de Oviedo en 2017 y cursando el Máster en Ingeniería Web en la misma universidad. Actualmente contratado como Ingeniero Informático en la empresa *Mecalux Software Solutions* (Gijón) (España). Contacto: uo206367@uniovi.es.

José Manuel Redondo Profesor Contratado Doctor en el Departamento de Informática de la Universidad de Oviedo, España. Ingeniero Técnico en Informática (2000), Ingeniero en Informática (2002) y Doctor en Informática (2007). Sus líneas de investigación son: lenguajes dinámicos, reflexión computacional, compilación JIT, máquinas virtuales y seguridad informática. Contacto: redondojose@uniovi.es.

Luís Vinuesa Profesor Colaborador en el Departamento de Informática de la Universidad de Oviedo, España. Ingeniero Técnico en Informática (1994), Ingeniero en Informática (1998) y Doctor en Informática (2007). Sus líneas de investigación son: lenguajes dinámicos, reflexión computacional, compilación JIT y orientación a aspectos. Contacto: vinuesa@uniovi.es.