Towards Lightweight Mobile Pentesting Tools to Quickly Assess Machine Security Levels

Iñigo Llaneza, Jose Manuel Redondo, and Luis Vinuesa

***Abstract*—Maintaining an appropriate security level on computing device infrastructures requires periodic audits to check for potential vulnerabilities. To do this, appropriate security tools are needed. Sometimes these tools turn difficult to deploy or use in scenarios where the infrastructure location, resource availability or general status do not suit their resource needs or features. This paper describes *Lightpen*, a security audit tool able to perform a fully customizable suite of lightweight security tests from a mobile device. *Lightpen* allows users to perform pentesting tasks at any place, choosing only the type of tests that are judged as the most adequate for the type of infrastructure to be audited. This way, resource and time usage is optimized, enabling security audits whenever they are needed. The tool is also very modular, using reflection to add more types of scans easily, without modifying the application core.**

***Index Terms*—Mobile device, customization, pentesting, lightweight, reflection**

I. Introducción

ACTUALMENTE es muy frecuente que los sistemas informáticos se vean expuestos a ataques, siendo uno de los problemas más complejos e importantes a los que se enfrentan los responsables de su seguridad. Cada año se incrementa la cantidad de ataques [1] de distintos tipos [2] que una máquina o infraestructura puede sufrir, afectando no solo a empresas [3] sino también a instituciones públicas, gobiernos [4] o infraestructuras militares [5]. Paralelamente al incremento del número de ataques también se agravan sus posibles consecuencias. Ejemplos comunes son los robos de información y revelación de secretos [6], interrupciones o denegaciones de servicio (DoS) [7] [8], secuestro de datos [9] o toma de control/aprovechamiento de infraestructuras para fines maliciosos/minado de criptomonedas (*cryptojacking*) [10] [11]. Hacer los sistemas informáticos más resistentes a ataques es por tanto una necesidad crítica en la actualidad [12].

Para comprobar la seguridad de los sistemas informáticos se audita su nivel de seguridad. Para ello, profesionales contratados aplican las mismas técnicas que los posibles atacantes, pero en un entorno controlado y bajo unas condiciones acordadas de antemano. El resultado de estas actividades determina la capacidad de una infraestructura de resistir ataques. La información obtenida por la auditoria se plasma en un informe donde se discuten los problemas encontrados, el nivel de exposición a ataques que suponen y sus posibles soluciones. Esta actividad se denomina *test de penetración* o *pentesting* [13], y a los auditores que la realizan se les suele denominar *pentesters*.

Los trabajos de *pentesting* normalmente constan de una parte más manual y detallada. En ella, el profesional pone a prueba el sistema usando determinadas técnicas junto con software especializado que permite el control total de lo que se va a ejecutar, como `NMap` [14] [15]. Esta labor suele complementarse con otra más automatizada, usando *herramientas de exploración automática de vulnerabilidades*. El software más avanzado de esta clase [16] [17] [18] [19] es muy complejo y tiene capacidades de detección muy avanzadas [20]. Estas herramientas suponen una ayuda inestimable, ya que permiten lanzar automáticamente una gran cantidad de pruebas sobre una infraestructura con muy poco esfuerzo, descubriendo potenciales puntos débiles. De esta forma, el *pentester* puede ejecutarlas en primera instancia para hacerse una idea aproximada de a dónde puede dirigir una investigación posterior más manual y minuciosa, en función de los errores más críticos que hayan identificado, incrementando la productividad y efectividad de la auditoría.

No obstante, desplegar una de estas herramientas rápidamente en un punto concreto de una infraestructura puede ser bastante costoso. Las más avanzadas requieren una gran cantidad de recursos computacionales y de red (`Nessus` por ejemplo recomienda usar 8Gb de RAM y una CPU de 4 cores a 2Ghz para funcionar correctamente [21]) y tiempo para realizar sus estudios, especialmente si la infraestructura a analizar es muy grande. Además, dado que el nº de test que implementan es muy grande, no siempre es sencillo o posible seleccionar con precisión cuáles se quieren ejecutar. Por ello, un *pentester* puede tener dificultades usándolas si solo necesitase un conjunto reducido de test que se adapten al tipo de sistema a analizar o a la información que ya posee sobre la infraestructura.

Por otro lado, hoy en día hay un auge tanto de los *dispositivos IoT* (*Internet of Things*) como de *dispositivos móviles* entre la población mundial. Cada vez es más frecuente [22] que dispositivos de tipos muy distintos (TVs, electrodomésticos, cámaras de vigilancia,...), usados para resolver problemas de requisitos muy diferentes y organizados usando distintas arquitecturas [23], estén conectados a Internet para cumplir con sus funciones. Respecto a los dispositivos móviles como *smartphones*, *tablets*, etc., cada vez es mayor el porcentaje de la población mundial que usa sus capacidades para lograr conectividad a sus servicios de Internet en cualquier momento y lugar [24]. En el contexto de este artículo, el principal problema de los dispositivos IoT es que se descubren con

Universidad de Oviedo, España, e-mail: uo206367@uniovi.es.
Universidad de Oviedo, España, e-mail: redondojose@uniovi.es (Corresponding author).
Universidad de Oviedo, España, e-mail: vinuesa@uniovi.es.
Manuscript received April 19, 2018; revised August 26, 2018.

bastante frecuencia casos de fallos de seguridad graves en los mismos, debido fundamentalmente a malas configuraciones, falta de actualizaciones o de soporte por sus fabricantes [25] [26]. La enorme presencia que tienen estos dispositivos actualmente, combinada con la elevada conectividad que dan los dispositivos móviles, incrementa aún más la superficie de ataque de una infraestructura de una empresa que los use y los potenciales peligros para los usuarios de las mismas.

Esta situación plantea dos problemas principales a los que hay que buscar una solución satisfactoria:

1) Hay ocasiones en que la seguridad del objetivo es muy baja, y en una auditoría es posible detectar problemas de seguridad muy graves mediante el examen rápido de puntos muy concretos. Esto hace que el resto de los problemas que puedan encontrarse no sea relevante: el sistema necesita una reconstrucción o mejora de base ya que parte de unas premisas erróneas, y por tanto lanzar un escaneo completo o sin la granularidad adecuada con una herramienta automática como las mencionadas supone un esfuerzo de tiempo y recursos innecesario. Ejemplo de este tipo de situaciones son los servicios ofrecidos por sistemas operativos sin soporte del fabricante (por ejemplo, `Windows XP`), o bien por versiones antiguas y con errores de seguridad graves conocidos y documentados de un software [27]. Detectar errores de configuración o fugas de información no tiene relevancia si el software en sí debe ser reemplazado completamente. Si el *pentester* no tiene la precaución de hacer un estudio exploratorio preliminar del sistema, eligiendo una serie de test particulares (algo que podría ser difícil de hacer en las herramientas mencionadas), puede invertir demasiado tiempo en llegar a la conclusión de que el sistema es inseguro de base y, por tanto, solo cabe su reconstrucción.
2) Es frecuente encontrar *dispositivos IoT* vinculados a redes `Wifi` ad-hoc, situadas en lugares estratégicos con cobertura limitada. Esto hace que no siempre sea posible o sencillo introducir un dispositivo con la potencia de cómputo y software de auditoría de seguridad adecuados para realizar auditorias de seguridad. La red puede no estar dimensionada para soportar el tráfico generado por este tipo de herramientas, o bien no es posible encontrar un punto centralizado y accesible desde el exterior para examinar todos los dispositivos de esta clase eficientemente, dada la disposición de estos y los posibles problemas de movilidad o conectividad con la infraestructura estudiada de la máquina que contiene el software de auditoría.

Por todo esto, el objetivo de la investigación presentada en este artículo es determinar si es posible usar la capacidad de cómputo y recursos de conectividad de un dispositivo móvil actual para tratar de solventar los problemas vistos. Para ello queremos demostrar que un dispositivo móvil tiene los recursos suficientes para hacer test de seguridad cuyo coste computacional y de ancho de banda sea relativamente bajo. Otro objetivo principal es permitir la personalización completa de los test a realizar, pudiendo ejecutar solo aquellos test que el usuario realmente quiera y por tanto reduciendo el consumo de recursos al mínimo posible. Esto permitiría al auditor determinar rápidamente el nivel de seguridad inicial de una infraestructura, y por tanto si merece la pena seguir haciendo un estudio más profundo de la misma. En caso de que se decida continuar, también puede servir para orientar la auditoría hacia aquellos aspectos concretos (máquinas, servicios,...) que merece la pena estudiar primero por mostrar indicios de un nivel de seguridad menor. Usar un dispositivo móvil permitirá además al usuario hacer la auditoría en cualquier localización rápidamente y agregarse a cualquier red `Wifi` bajo demanda, examinando rápidamente cualquier dispositivo independientemente de su localización. De esta forma, queremos demostrar que es posible conseguir una herramienta de auditoría de seguridad de movilidad muy alta, capaz de ejecutar test de seguridad concretos, personalizados, rápidos y adecuados a dispositivos de baja capacidad computacional y que, gracias a su movilidad y bajo consumo de recursos, permita dar respuesta a los problemas vistos.

El resto del artículo se estructura de la siguiente forma: la sección II describe cómo hemos planteado la resolución del problema mediante la creación de una aplicación móvil que responda a las premisas enunciadas, mientras que la sección III describe la arquitectura de cada una de sus partes. La sección IV explica el proceso de ejecución del prototipo actual de la misma y los usos que se le están dando en la actualidad, mientras que la sección V detalla las conclusiones de nuestra investigación y el trabajo futuro proyectado. Finalmente, la sección VI hace una descripción de una serie de trabajos relacionados con nuestra propuesta.

II. Planteamiento del problema

La primera decisión tomada a la hora de plantear soluciones a los problemas vistos es relativa a la movilidad. Para que la herramienta de auditoría cumpla con sus objetivos, no solo tiene que ajustar el conjunto de test a aquellos que sean abordables por la capacidad de cómputo de un dispositivo móvil, sino que debe ejecutarse sobre el mayor número posible de dispositivos para que sea más fácil introducirla en cualquier entorno. En ese sentido la elección de plataforma de desarrollo es clara: actualmente, el sistema `Android` está presente en el 84,8% de dispositivos móviles a nivel mundial [28].

El segundo problema a resolver es relativo a los test de seguridad. Más allá de la carga computacional que tengan, lo importante es que estos tengan una identidad propia que les permita ser seleccionados de forma independiente a los demás. Cada test tendrá un fin concreto y claro dentro de la herramienta, y será seleccionable individualmente para poder formar perfiles de escaneo con un conjunto de ellos. Por tanto, es importante delimitar las funciones de los test de manera muy clara, y presentarlos en forma de unidades de código autoexplicativas y autocontenidas. Para ello se ha decidido que lo mejor es optar por un sistema de *plugins* de concepto similar al de herramientas como `ZAP` [18] o `w3af` [29].

Además de la granularidad de los plugins, es importante también plantear la variedad de los mismos. Dado que la herramienta a diseñar es un prototipo demostrativo del objetivo

de investigación de este artículo, el número inicial de *plugins* que diseñaremos será limitado. No obstante, la herramienta debe tener capacidad de crecimiento para incorporar nuevos test de forma sencilla, ya no solo por una posible comercialización futura, sino para permitir que se pueda personalizar de acuerdo a las necesidades de usuarios avanzados. Esto nos lleva a dos decisiones a la hora de abordar el problema: (1) que la herramienta sea *open source* de cara a que cualquier usuario pueda auditar, modificar o ampliar sus funcionalidades, y (2) que la carga de *plugins* se haga bajo demanda mediante un protocolo bien definido. De este modo, la estructura de la aplicación a diseñar debe constar de un núcleo de funcionalidades comunes a todos los *plugins* y un cargador de *plugins* que permita, mediante reflexión computacional, cargar dinámicamente *plugins* que se localicen en rutas predeterminadas y que cumplan con una interfaz de servicios que los caracterice, asegurándose así la correcta comunicación de la aplicación con cualquier futuro *plugin* diseñado para la misma. Esto además tiene la ventaja de cargar solo aquellos que realmente van a usarse en cada momento, consumiendo menos recursos que una estructura monolítica y siendo así más adecuado para las plataformas móviles que lo van a ejecutar.

Finalmente, dado que el objetivo de la aplicación es hacer labores de *pentesting*, el resultado de la ejecución de los *plugins* elegidos es un informe que detalle las vulnerabilidades encontradas, como se mencionó en la sección I. Para usar un formato de información que pueda ser contrastada con la obtenida de otras fuentes, toda la información sobre vulnerabilidades presentada por la aplicación se hará de acuerdo con los estándares y clasificación OWASP [30].

Con estos planteamientos, en la siguiente sección pasaremos a detallar la arquitectura de *Lightpen*, la herramienta que hemos diseñado para probar la viabilidad de la investigación planteada en este artículo.

III. Solución propuesta

Para lograr los objetivos de consumir muy pocos recursos y permitir al usuario seleccionar las pruebas que desee en cada momento, el diseño de *Lightpen* consta de varios módulos. La Fig. 1 muestra la arquitectura general de la aplicación.

- *Cargador de plugins* (`PluginLoader`): Para cumplir con lo establecido en la sección anterior, la parte más importante de la aplicación es un cargador dinámico de los *plugins* que modelan cada test de seguridad, ocupándose también de gestionarlos y lanzarlos coordinadamente. Además, recogerá los resultados de su ejecución para suministrárselos al módulo encargado de presentarla. Este cargador se ha diseñado intentando maximizar su rendimiento, de forma que la mayor parte de la carga computacional de la aplicación recae en la ejecución de los *plugins* cargados y seleccionados por el usuario.
- *Plugins con test de seguridad*: Cada uno de los test de seguridad ejecutados por la aplicación estará contenido en su *plugin* de seguridad correspondiente (ver la subsección III-B), configurando sus opciones en función de las preferencias del usuario suministradas por el cargador de *plugins*. Cada *plugin* deberá ser capaz de ejecutarse cuando el cargador se lo indique, devolviéndole la información obtenida una vez procesada.
- *Operaciones comunes para plugins*: Aunque cada *plugin* implementa un test de seguridad distinto (ver sección II), para su implementación es posible que se necesiten usar determinadas operaciones que otros *plugins* también pueden necesitar. Estas operaciones se ofrecen en forma de API para ahorrar trabajo a los implementadores de *plugins* y lograr una base de desarrollo común a todos ellos, de forma que sea más sencilla su depuración y optimización.

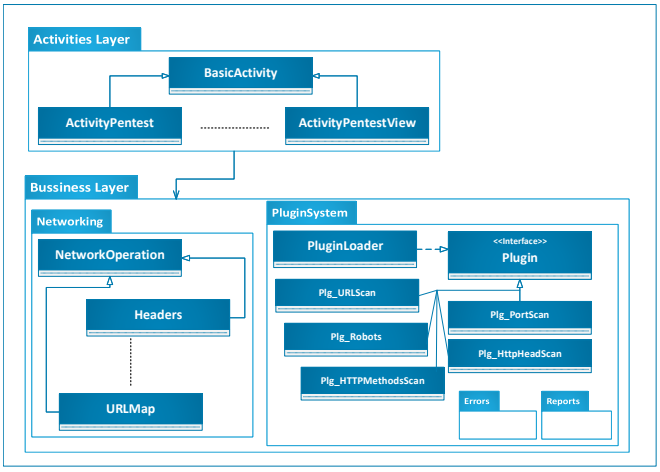

Fig. 1. Arquitectura de *Lightpen*.

A continuación, describiremos más en detalle las partes más importantes de la arquitectura del sistema.

A. Cargador de Plugins

El cargador de plugins posibilita que *Lightpen* sea una aplicación altamente modular, al separar la gestión de *plugins* de las operaciones que hacen los mismos en bloques independientes. Esto hace que *Lightpen* sea una aplicación adaptativa en tiempo de ejecución. Por otro lado, se facilita la evolución y personalización de la aplicación a través de la construcción de nuevos *plugins* y también su mantenimiento (mediante la modificación o corrección de aquellos plugins que presenten errores o necesiten mejoras) sin cambiar el código de la aplicación principal.

Para lograr esto se han usado las capacidades de introspección [31] del lenguaje *Java*, a través de las funcionalidades del paquete `java.lang.reflect`. La carga de nuevos *plugins* es dinámica gracias al uso del método `Class.forName(nombreDeLaClase:String)`, que cargará el *plugin* cuyo nombre le indiquemos como parámetro. Para que esto sea posible, el diseño de *Lightpen* especifica que todo plugin susceptible de ser cargado dinámicamente debe residir en una ruta concreta, configurable mediante los ficheros de configuración de la aplicación.

B. Estructura de los Plugin y normas para su creación

Como se mencionó en la sección II, *Lightpen* facilita la creación de nuevos *plugins* siguiendo una serie de normas de

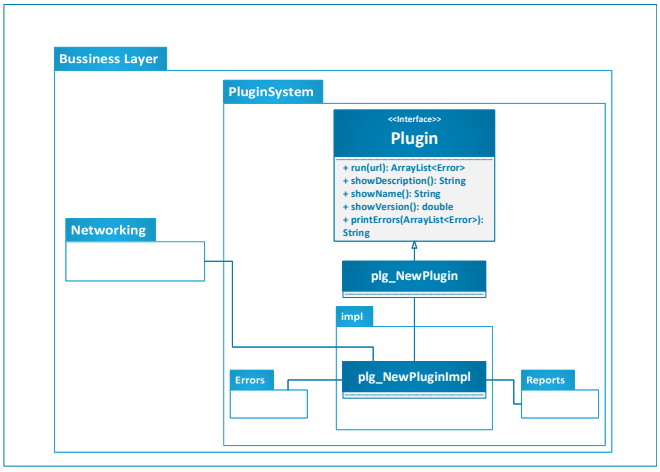

Fig. 2. Estructura de un plugin.

programación. Toda clase que las cumpla y que se encuentre en la ruta mencionada anteriormente se cargará como *plugin* utilizable por la aplicación, lo cual hace que su número pueda crecer dinámicamente de forma sencilla. Estas normas son:

1) *Que la clase implemente la interfaz* `Plugin`: Esta interfaz contiene una serie de servicios básicos que la aplicación solicitará al *plugin* en determinados momentos, y que éste debe implementar para que la interacción con la aplicación sea correcta. La Figura 2 muestra la API de servicios a implementar.
2) *Que el nombre de la clase comience por el prefijo* `plg_` para distinguir las clases principales que implementan la funcionalidad de los *plugins* de otras posibles clases auxiliares que sean necesarias.

La estructura completa de la implementación de un nuevo *plugin* se muestra en la Figura 2. Actualmente, los *plugins* implementados en la herramienta separan su interfaz de servicios (`plg_NewPlugin`) de la implementación de los mismos (`plg_NewPluginImpl`), que a efectos de arquitectura de la aplicación se sitúa en un paquete aparte. Esta separación no es obligatoria a la hora de desarrollar nuevos *plugins*. La implementación será la encargada de modelar la funcionalidad concreta de cada *plugin* cumpliendo con la interfaz de servicios. Los paquetes `networking`, `errors` y `reports` son el API de servicios comunes reutilizable por los *plugins* mencionado, y que detallaremos en la subsección III-C.

C. API de operaciones comunes para el desarrollo de plugins

La API de operaciones comunes para *plugins* contiene principalmente operaciones de red (paquete `networking`), así como otras funcionalidades auxiliares relativas a la gestión de errores (paquete `errors`) y mostrar los resultados de las operaciones (paquete `reports`). A modo de ejemplo, la Tabla I describe las operaciones principales disponibles en la API de operaciones de red y su función. Algunas de estas operaciones pueden verse en el diagrama de la Figura 1, donde también se muestra que todas ellas derivan de la clase `NetworkOperation`.

TABLE I
OPERACIONES PRINCIPALES DEL API PARA EL DESARROLLO DE PLUGINS.

Operación	**Descripción**
Obtener cabeceras HTTP (`Headers`)	Obtiene las cabeceras de una petición HTTP
Obtener HTML	Dada una conexión abierta, obtiene como `String` el código HTML de una web
Obtener métodos HTTP	Obtiene los métodos HTTP soportados por un servidor remoto
Crawler de URLs (`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
Escaneo de puertos	Escáner de servicios que mediante el uso de *sockets* rastreará posibles elementos activos

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 II, los *plugins* se han diseñado con la intención de que las operaciones que realicen sean autocontenidas, necesiten pocos recursos y terminen su ejecución en poco tiempo, adaptándose así a las posibilidades de un dispositivo móvil. La siguiente lista muestra los *plugins* por defecto incluidos en el prototipo inicial de *Lightpen*.

1) `HTTPHeadScan`: Analiza las cabeceras de respuesta HTTP para determinar si se han activado determinados aspectos de seguridad (uso de `HSTS`, `HKPK`, habilitada la protección `XSS`,...).
2) `RobotScan`: Determina si existe un fichero `robots.txt` y analiza su contenido para ver si proporciona información que facilite la ejecución de algún ataque.
3) `HTTPUrlScan`: Crea un mapa de los recursos asociados a una URL y permite visualizarlos para comprobar que no se está enlazando públicamente ninguna URL no deseada.
4) `HTTPMethodScan`: Analiza los métodos HTTP soportados por un servidor, para localizar aquellos que puedan suponer un problema de seguridad.
5) `HTTPPortScan`: Analiza los puertos de un servidor en busca de servicios en ejecución que puedan comprometer la seguridad del mismo.

Por otro lado, los siguientes *plugins* se encuentran actualmente en fase de planificación y desarrollo para ser incorporados en las siguientes versiones de la aplicación:

1) `HTTPProductScan`: Analiza cabeceras HTTP para buscar nombres de productos software. Incluye además una búsqueda automatizada de `CVEs` [32] que describan vulnerabilidades de esos productos, para hacer un per-

fil de vulnerabilidades del servidor en función de los productos de terceros que use.

2) `BlacklistScan`: Comprueba si la IP o URLs asociadas a la máquina remota figura en alguna lista negra conocida como *Google Safe Browsing* [33], 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` [14].
3) `PhisingDomainScan`: Determina si existen nombres de dominio similares a los asociados con el 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) `FileTipsScan`: 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 examinará la presencia de dichos ficheros o rutas conocidas en la máquina remota de acuerdo con las especificaciones de su fabricante. Se trataría por tanto de una implementación de la técnica *static files* de herramientas como `BlindElephant` [34].
5) `ErrorPageScan`: Comprueba la existencia de páginas de error no personalizadas, que puedan indicar el tipo o versión de algún producto.
6) `DirListingScan`: Comprueba si el servidor tiene habilitado el listado de directorios cuando se accede a una ruta del mismo sin especificar un fichero concreto, lo cual es un problema de seguridad al estar dando demasiada información acerca de los contenidos del mismo.
7) `CommentScan`: Analiza los comentarios presentes en las webs o ficheros que sirve el sistema remoto, para buscar palabras clave o comentarios por defecto que puedan delatar el uso de determinados productos de terceros y/o versiones de los mismos.
8) `MetaScan`: Analiza la presencia de metadatos que puedan dar demasiada información en los ficheros o imágenes servidos por el sistema remoto.

Para lograr un diseño de plugins donde sus fines sean claros y distinguibles del resto, como se específico en la sección II, todos los *plugins* contenidos y planificados en *Lightpen* están asociados a alguna posible vulnerabilidad clasificada por OWASP. La Tabla II muestra esta asociación.

TABLE II
CLASIFICACIÓN OWASP DE LAS VULNERABILIDADES CUBIERTAS POR *Lightpen*

Nombre del plugin	Vulnerabilidad(es) OWASP
`HTTPHeadScan`	*Test HTTP Strict Transport Security* (`OTG-CONFIG-007`) *Testing for Reflected Cross Site Scripting* (`OTG-INPVAL-001`) *Testing for Stored Cross Site Scripting* (`OTG-INPVAL-002`)
`RobotScan`	*Review Webserver Metafiles for Information Leakage* (`OTG-INFO-003`)
`HTTPUrlScan`	*Map execution paths through application* (`OTG-INFO-007`) *Enumerate Applications on Webserver* (`OTG-INFO-004`) *Identify application entry points* (`OTG-INFO-006`) *Map Application Architecture* (`OTG-INFO-010`)
`HTTPMethodScan`	*Test HTTP Methods* (`OTG-CONFIG-006`)
`HTTPPortScan`	*Enumerate Applications on the Webserver* (`OTG-INFO-004`) *Identify application entry points* (`OTG-INFO-006`)
`HTTPProductScan`	*Enumerate Applications on Webserver* (`OTG-INFO-004`) *Fingerprint Web Application Framework* (`OTG-INFO-008`) *Fingerprint Web Application* (`OTG-INFO-009`)
`FileTipsScan`	*Test File Extensions Handling for Sensitive Information* (`OTG-CONFIG-003`) *Review Old, Backup and Unreferenced Files for Sensitive Information* (`OTG-CONFIG-004`)
`ErrorPageScan`	*Testing for Error Code* (`OTG-ERR-001`)
`DirListingScan`	*Test File Extensions Handling for Sensitive Information* (`OTG-CONFIG-003`) *Review Old, Backup and Unreferenced Files for Sensitive Information* (`OTG-CONFIG-004`)
`CommentScan`	*Review Webpage Comments and Metadata for Information Leakage* (`OTG-INFO-005`)
`MetaScan`	*Review Webpage Comments and Metadata for Information Leakage* (`OTG-INFO-005`)

IV. RESULTADOS

Actualmente hemos implementado un prototipo funcional de *Lightpen* para `Android Marshmallow` (6.0, API Level 23) o superior, presente en la actualidad en un 64% de los dispositivos `Android` [35]. El primer prototipo ocupa 1.3Mb e implementa los *plugins* mencionados en la sección III. La aplicación se ha ejecutado satisfactoriamente sobre *smartphones* modernos de varias gamas: `Nexus 5` (año 2013, CPU `Qualcomm Snapdragon 800`, 4 núcleos a 2.3Ghz, 2Gb RAM), `Samsung Galaxy S3` (año 2012, CPU `Samsung Exynos 4412`, 4 núcleos a 1.4Ghz, 1Gb de RAM), `Sony Xperia Z` (año 2013, `CPU Qualcomm APQ8064`, 4 núcleos a 1.5Ghz, 2Gb RAM) y `Sony Xperia Z5` (año 2015, CPU `Qualcomm Snapdragon 810`, 8 núcleos a 2Ghz, 3Gb RAM).

El proceso para lanzar un escaneo de vulnerabilidades sobre un sistema remoto es muy sencillo. La Figura 3 muestra la pantalla principal de *Lightpen*. En ella el usuario puede acceder a la funcionalidad de escaneo o ver el resultado de escaneos anteriores. Una vez seleccionada la opción de escaneo, *Lightpen* cargará los *plugins* disponibles en ese momento en la instalación local y le dará al usuario la posibilidad de

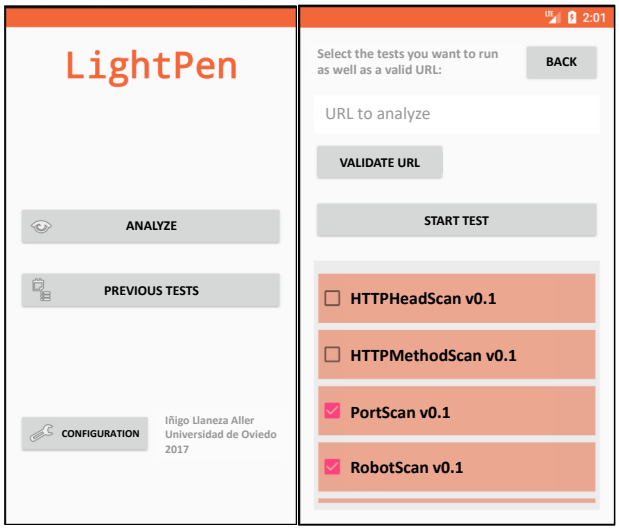

Fig. 3. Proceso para lanzar un escaneo en *Lightpen*.

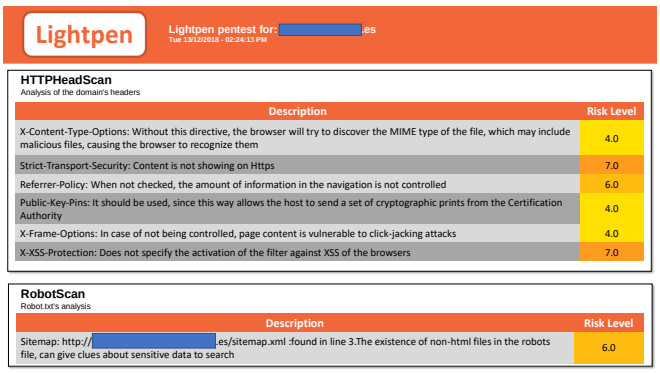

Fig. 4. Resultados de un escaneo en *Lightpen*.

seleccionar los que desee. Además, deberá indicar la dirección del sistema remoto a analizar, con la opción de analizar si dicha dirección se encuentra accesible antes de ejecutar el escaneo. Una vez hecho esto, el sistema pasará a ejecutar los *plugins* seleccionados.

Al terminar el escaneo el usuario podrá ver el resultado del mismo. El prototipo actual solo permite por el momento mostrar los resultados en formato HTML estándar, de manera que puedan ser visualizados en cualquier navegador. 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 la importancia de la misma se calcula de la misma forma que lo hace OWASP [36]. La aplicación también permite compartir esos resultados con otras aplicaciones que acepten el tipo `MIME text/html`, usando los mecanismos para compartir ficheros estándar de `Android`.

Lightpen ha sido usada contra servidores de empresas reales bajo petición de sus interesados, así como contra servidores de la *Escuela de Ingeniería Informática* de la *Universidad de Oviedo* [37], también bajo petición expresa de sus dueños y conectándose a su red `Wifi`. *Lightpen* fue utilizada como paso previo a una auditoría de seguridad completa contra servidores de la empresa *Mecalux Software Solutions* [38] por los empleados de la empresa encargados de realizarla. Durante ese proceso, gracias a *Lightpen* pudieron detectarse la mayoría de vulnerabilidades existentes relativos a errores con las cabeceras HTTP del servidor web, errores de configuración, servicios y puertos abiertos de forma indebida. Esto hizo que la auditoría completa posterior fuese más dirigida y pudiese finalizarse en menor tiempo del inicialmente previsto, por lo que la herramienta cumplió con el propósito para el que fue diseñada.

V. Conclusiones

Lightpen es una herramienta altamente modular que permite hacer rápidamente test de seguridad personalizados de bajo consumo de recursos computacionales sobre equipos remotos en cualquier momento y lugar. El conjunto de test que realiza es completamente configurable, y la información devuelta por la aplicación puede extenderse o personalizarse gracias a que puede enviarse a otras aplicaciones para su procesamiento. Con esta herramienta, cualquier *pentester* podrá obtener información preliminar acerca del nivel de seguridad de un equipo remoto bajo demanda y desde su propio dispositivo móvil. Esta información puede resultar muy valiosa para dirigir futuras operaciones de *pentesting* en profundidad o para identificar rápidamente máquinas muy vulnerables, y así tomar acciones inmediatas para mitigar cualquier posible problema de seguridad que pudieran causar a una infraestructura. 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 en tareas de *pentesting*, alcanzando los objetivos planteados. Por tanto, usar un dispositivo móvil para ejecutar tareas de auditoría de seguridad en las condiciones planteadas es una línea de trabajo viable en la que se va a seguir investigando.

En este sentido el trabajo futuro consistirá precisamente en incidir más en el carácter móvil de la aplicación, haciendo la presencia de una herramienta como *Lightpen* más permanente en una infraestructura. De esta forma, en lugar de hacer escaneos bajo demanda, se trataría de comprobar si es posible crear un dispositivo de vigilancia periódica de una red de dispositivos heterogéneos, pensando especialmente en dispositivos IoT. Dado que dichos dispositivos presentan un número no despreciable de vulnerabilidades por los motivos comentados anteriormente, una herramienta que tenga el mismo carácter que dichos dispositivos, pero cuya función sea la vigilancia y auditoría de toda la infraestructura, puede ser clave a la hora de detectar a tiempo un posible ataque de seguridad. Por tanto, la próxima iteración de esta línea de investigación es crear una evolución de *Lightpen* que tenga algunas características de IDS (*Intrusion Detection System*), estudiando también la viabilidad de que pueda anular la comunicación de aquellos dispositivos en los que se detecte algún problema de seguridad que comprometa toda la infraestructura, aislando así posibles fuentes de peligro para la misma.

Otras líneas de trabajo futuras son incorporar todos los *plugins* descritos en la sección III al prototipo existente para

aumentar la funcionalidad de la herramienta. Además, una vez que el conjunto de *plugins* sea más amplio, se plantea agrupar varios de ellos en *perfiles de escaneo*, que permitan la selección sencilla de múltiple *plugins* afines para hacer escaneos predeterminados más rápidamente. Otra línea de trabajo es estudiar qué formatos de informes adicionales pueden ser más útiles de cara a integrarlos con otras herramientas existentes. Por otro lado, se está contemplando la posibilidad de suministrar esta herramienta a los alumnos de asignaturas de seguridad y administración de servidores [39] del *Grado de Ingeniería Informática* [37] y del *Máster en Ingeniería Web* [40] de la *Universidad de Oviedo*, para que los alumnos sean conscientes de la importancia de las tareas de descubrimiento de información a la hora de analizar la seguridad de un equipo remoto. Además, también se está estudiando la posibilidad de usar la herramienta contra objetivos más grandes, como webs realistas que presentan vulnerabilidades de seguridad introducidas a propósito [41], también para facilitar el aprendizaje de técnicas de seguridad, o web reales en producción desarrolladas anteriormente por miembros de nuestro grupo de investigación [42], previa autorización de sus propietarios.

La versión actual de *Lightpen* y su conjunto de *plugins* incluidos más actual puede descargarse de https://github.com/LlanezaAller/Lightpen.

VI. Trabajo relacionado

Como mencionamos en la sección I, existen una serie de herramientas de análisis automático de vulnerabilidades que permiten a un *pentester* obtener mucha información acerca de los diferentes problemas de seguridad que pueden tener equipos remotos. Algunas de ellas hacen estudios exhaustivos, pero a costa de requerir muchos recursos computacionales y tiempo para realizarlos, además de no permitir una personalización muy granular de los test que ejecutan. Ejemplos de las mismas son *OpenVAS* [16] y *Nessus* [17]. Debido a ello, no son adecuadas para poder ejecutarse desde plataformas móviles ni por tanto son fácilmente portables a localizaciones donde sean más necesarias. En esta categoría también podemos incluir el módulo de descubrimiento de vulnerabilidades de la conocida herramienta *Metasploit* [43], aunque esta herramienta es más conocida por su habilidad para explotar vulnerabilidades.

Existen otra serie de herramientas cuya labor está especializada en análisis de servidores web. Algunas de ellas, como *Nikto* [44], tienen un consumo de recursos muy bajo, pero no permiten personalizar con mucha granularidad las opciones de sus análisis. Otra herramienta en esta categoría es *OWASP ZAP* [18], de un consumo más alto de recursos que la anterior y por tanto inadecuada para plataformas móviles, pero que también permite expandir sus funcionalidades gracias a la programación de módulos y otro tipo de mecanismos de extensión [45].

Por otro lado, la herramienta clásica de seguridad *NMap* [14] tiene un consumo de recursos muy bajo y por tanto puede ejecutarse desde plataformas móviles. Sin embargo, su gran nº de opciones hace que sea difícil personalizar el tipo de análisis de seguridad a realizar, al carecer de un interfaz gráfico que facilite la comprensión de sus test y la personalización individual de las diversas opciones que tienen. Esto hace que existan iniciativas para desarrollar GUIs más completos [15].

Existen otras herramientas de pentesting para `Android` similares a *Lightpen*. Algunas de ellas, como `Droidbug Pentesting` [46], son implementaciones de distribuciones *Linux* sobre `Android`, con múltiples herramientas de *pentesting* populares que deben usarse individualmente, y por tanto obligan al usuario a conocer el funcionamiento y parámetros de cada una de ellas. Otras están enfocadas solamente al *pentesting* web, como `Kayra the Pentester Lite` [47], un escáner de vulnerabilidades que incluye una variedad de test para descubrir vulnerabilidades en servidores HTTP y que permite enviar los resultados vía e-mail. Además, podemos mencionar `HTTP Tools` [48], 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 y las webs alojadas por los mismos.

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` [49].

Finalmente, podemos encontrar aplicaciones que incorporan funcionalidades que permiten explotar las vulnerabilidades de los sistemas que analizan, lo cual es un aspecto que no se ha contemplado en el diseño de *Lightpen* ya que su labor está orientada a la localización y prevención de las mismas. Ejemplos de este tipo de herramientas son `Info P. D. S.` [50] o `Andro Hackbar` [51], especializada en la inyección de ataques *XSS* o *SQL Injection*.

Acknowledgment

Este trabajo ha sido financiado por la Unión Europea a través del Fondo Europeo de Desarrollo Regional (FEDER), así como por el Principado de Asturias a través de su plan de innovación, tecnología y ciencia (GRUPIN-14-100). Los autores también han recibido financiación del Banco Santander a través del Campus de Excelencia Internacional.

References

[1] C. Mclellan, "Cybersecurity predictions for 2016: How are they doing?" www.zdnet.com/article/cybersecurity-predictions-for-2016-how-are-they-doing, 2016, (Dec 10, 2018).

[2] Rapid7, "Common types of cybersecurity attacks: A look inside the attacker's toolkit," www.rapid7.com/fundamentals/types-of-attacks, 2017, (Dec 10, 2018).

[3] M. Smith, "Huge rise in hack attacks as cyber-criminals target small businesses," www.theguardian.com/small-business-network/2016/feb/08/huge-rise-hack-attacks-cyber-criminals-target-small-businesses, 2016, (Dec 10, 2018).

[4] H. Security, "Rise of cyber attacks against the public sector," www.helpnetsecurity.com/2016/09/23/cyberattacks-public-sector/, 2016, (Dec 10, 2018).

[5] C. Pellerin, "Cybercom: Pace of cyberattacks have consequences for military, nation," www.defense.gov/News/Article/Article/1192583/cybercom-pace-of-cyberattacks-have-consequences-for-military-nation/, 2017, (Dec 10, 2018).

[6] T. Armerding, "The 16 biggest data breaches of the 21st century," www.csoonline.com/article/2130877/data-breach/the-16-biggest-data-breaches-of-the-21st-century.html, 2017, (Dec 10, 2018).
[7] T. Matthews, "What DDoS attacks really cost businesses," Imperva Incapsula, Tech. Rep., 2016, (Dec 10, 2018).
[8] 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.
[9] 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.
[10] I. Institute, "ICMP reverse shell," resources.infosecinstitute.com/icmp-reverse-shell/#gref, 2014, (Dec 10, 2018).
[11] 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.
[12] Checkpoint, "Live cyber attack threat map," threatmap.checkpoint.com/ThreatPortal/livemap.html, 2018, (Dec 10, 2018).
[13] OWASP, "Penetration testing methodologies," www.owasp.org/index.php/Penetration_testing_methodologies, 2016, (Dec 10, 2018).
[14] NMap.org, "Nmap: The network mapper," nmap.org/, 2018, (Dec 10, 2018).
[15] D. Cuesta, "NMapGUI: Advanced graphical user interface for NMap," https://github.com/danicuestasuarez/NMapGUI, 2018, (Dec 10, 2018).
[16] Openvas.org, "OpenVAS: Open vulnerability assessment system," www.openvas.org/, 2018, (Dec 10, 2018).
[17] Tenable, "Nessus vulnerability scanner," www.tenable.com/products/nessus-vulnerability-scanner, 2018, (Dec 10, 2018).
[18] OWASP, "Zed attack proxy project," www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project, 2018, (Dec 10, 2018).
[19] ——, "Vulnerability scanning tools," https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools, 2018, (Dec 10, 2018).
[20] S. Chen, "Evaluation of web application vuln. scanners in modern pentest/SSDLC usage scenarios," http://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html, 2018, (Dec 10, 2018).
[21] Tenable, "Tenable Nessus scanner system requirements," https://docs.tenable.com/nessus/Content/HardwareRequirements.htm, 2018, (Dec 10, 2018).
[22] Statista.com, "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)," https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2018, (Dec 10, 2018).
[23] 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.
[24] B. Popper, "Google announces over 2 billion monthly active devices on Android," https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users, 2017, (Dec 10, 2018).
[25] 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.
[26] T. Spring, "IoT insecurity: Pinpointing the problems," https://threatpost.com/iot-insecurity-pinpointing-the-problems/119389/2/, 2016, (Dec 10, 2018).
[27] Mitre.org, "CVE details: The ultimate security vulnerability datasource," https://www.cvedetails.com/, 2018, (Dec 10, 2018).
[28] idc.com, "Smartphone OS market share," https://www.idc.com/promo/smartphone-market-share/os, 2018, (Dec 10, 2018).
[29] w3af.org, "w3af – open source web application security scanner," http://w3af.org/, 2018, (Dec 10, 2018).
[30] OWASP, *OWASP Pentesting Guide v4*. OWASP, 2017, vol. 1.
[31] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167642309000689
[32] Mitre.org, "Common Vulnerabilities and Exposures," cve.mitre.org/, 2018, (Dec 10, 2018).
[33] Google, "Google safe browsing," https://safebrowsing.google.com/, 2018, (Dec 10, 2018).
[34] P. Thomas, "BlindElephant web application fingerprinter," http://blindelephant.sourceforge.net/, 2018, (Dec 10, 2018).
[35] B. Stolyar and J. Chokkattu, "Android 8.0 Oreo operating system is now on 19 percent of active devices," https://www.digitaltrends.com/mobile/android-distribution-news/, 2018, (Dec 10, 2018).
[36] OWASP, "OWASP risk rating methodology," www.owasp.org/index.php/OWASP_Risk_Rating_Methodology, 2016, (Dec 10, 2018).
[37] E. de Ingeniería Informática, "Web oficial de la Escuela de Ingeniería Informática de la Universidad de Oviedo," ingenieriainformatica.uniovi.es/, 2018, (Dec 10, 2018).
[38] Mecalux, "Página oficial de Mecalux Esmena," https://www.mecalux.es/, 2018, (Dec 10, 2018).
[39] J. M. Redondo, "Improving student assessment of a server administration course promoting flexibility and competitiveness," *IEEE Transactions on Education*, pp. 1–8, 2018.
[40] E. de Ingeniería Informática, "Web oficial del Máster en Ingeniería Web de la Universidad de Oviedo," miw.uniovi.es/, 2018, (Dec 10, 2018).
[41] 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.
[42] 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.
[43] Rapid7, "Metasploit: The world's most used penetration testing framework," www.metasploit.com/, 2018, (Dec 10, 2018).
[44] Cirt.Net, "Nikto," cirt.net/Nikto2, 2018, (Dec 10, 2018).
[45] GitHub.com, "ZAProxy DevExtending," github.com/zaproxy/zaproxy/wiki/DevExtending, 2015, (Dec 10, 2018).
[46] BugsecApps, "Droidbug pentesting," play.google.com/store/apps/details?id=com.droidbugfree.es, 2018, (Dec 10, 2018).
[47] F. Caddesi and Y. Mahallesi, "Kayra the pentester lite," play.google.com/store/apps/details?id=teycode.kayralite, 2018, (Dec 10, 2018).
[48] CafeLabs, "HTTP tools," play.google.com/store/apps/details?id=com.cafelabs.curlme, 2018, (Dec 10, 2018).
[49] R. Kumar, "Hackode," play.google.com/store/apps/details?id=com.techfond.hackode, 2018, (Dec 10, 2018).
[50] V. Padovan, "Info P. D. S." play.google.com/store/apps/details?id=com.infopdshacker2015_wp, 2018, (Dec 10, 2018).
[51] Z. Developers, "Andro hackbar," play.google.com/store/apps/details?id=zyberph.hackbar.zk, 2018, (Dec 10, 2018).

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.