SIO FORENSICS

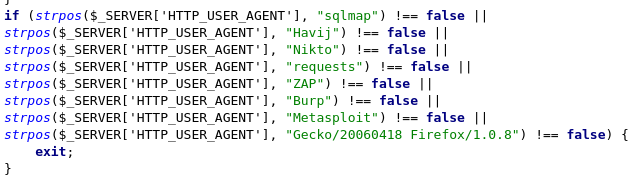
Mecanismos de confinamento encontrados

1. Uso de máquina virtual

Por si só, o recurso a uma máquina virtual para alocação de um servidor já pode ser considerado um mecanismo de confinamento, tendo em conta promover a independência relativamente ao hardware, maior facilidade na execução de backups e, sobretudo, criando uma sensação de isolamento relativamente a outros ambientes (sejam virtualizados ou físicos), criando assim um domínio de segurança apenas para um conjunto restrito de aplicações independente de uma máquina física.

1. Código “anti-penetração”

No ficheiro header.php aparece o seguinte código PHP:



Ora, como se pode observar, este código PHP previne que, se alguma ferramenta para testes de penetração e exposição de vulnerabilidades for utilizada, o sistema dá “exit”, impossibilitando essa mesma análise. Assim, dir-se-ia que este é um dos métodos de confinamento utilizados.

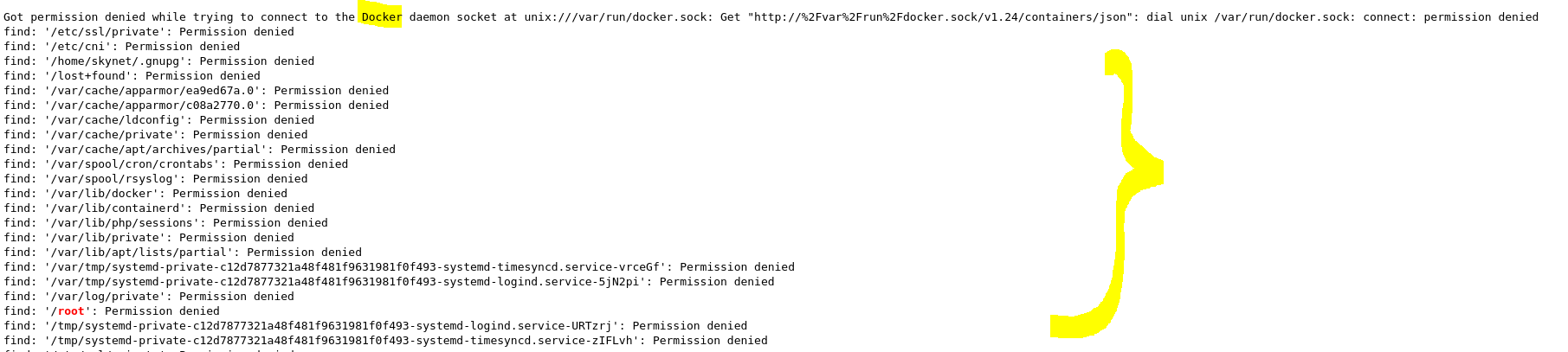
1. Implementação de Apparmor

Outro mecanismo de confinamento é o recurso ao Apparmor, um sistema de controlo de acessos com limitações das aplicações; podemos verificar a sua existência ao executar o comando **find**:



1. Uso de um utilizador com poucas permissões

(mais detalhes na secção de vulnerabilidades barradas)

Resumindo, o atacante tentou executar comandos e obter o mapeamento de ficheiros, mas não obteve permissões para tal; ora, tudo isto indica que (e como se comprovou) que por default ele está a usar um utilizador com poucas permissões de acesso, o que se pode comprovar ao analisar o log de erros:

Vulnerabilidades encontradas

1. Campos de login vulneráveis a SQL Injection e falta de verificações

Tentativa 1

Em primeiro lugar, o atacante tentou colocar uma pelica no campo de usermail do login, e apareceu o seguinte erro:

DB Error, could not query the database

MySQL Error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''''' at line 1

Ora, isto significa que o campo de login é vulnerável a SQL Injection!

O erro tem 5 pelicas seguidas, e podemos interpretar desta forma: ''' do campo do usermail, ou seja, início de string, 0 chars, fim de string, e início de string (sem fim); depois ainda tens mais 2 pelicas provenientes do campo da password (vazia) - 3 + 2 = 5.

Analisando igualmente o ficheiro login.php, em que ele primeiro verifica na base de dados se existe um username igual ao usermail introduzido, deduz-se que, por não haver um usermail igual a “vazio”, tenha então aparecido o erro no whireshark.

Tentativa 2

Desta vez, o atacante tentou colocar ' -- // no input reservado ao usermail; desta vez não deu nenhum erro de DB ou MySQL mas apareceu uma mensagem de “Invalid username, try again”.

Neste caso, ele ao colocar um comentário vai anular a componente do query destinada à password, pelo que é compreensível que o erro só se queixe do username/usermail; com a pelica introduzida no início ele fechou o query do usermail, e recebeu como "input" 0 chars tal como na primeira tentativa, mas como também foi colocado comentário na palavra-passe e por isso o campo da password foi ignorado e o query do usermail foi "fechado", então não houve erro de SQL Syntax.

O erro de username inválido (e não de sintaxe) deve ter aparecido por causa das verificações de html que são feitas do lado do cliente – o campo do usermail é do tipo “email”, logo é óbvio que um email igual a “vazio” é um email inválido.

Tentativa 3

Agora, ele tentou colocar no usermail ' OR 1=1 -- // e conseguiu entrar no sistema como admin, mesmo sem nenhuma palavra-passe.

Tal como na tentativa anterior, ele comentou o campo da password e, por isso, a parte do query destinada à palavra-passe vai ser novamente ignorada.

Para obter "dados", isto é, uma correspondência de uma conta na base de dados, alguma parte na query tem de dar valor lógico verdadeiro obrigatoriamente! ora, nada melhor do que garantir uma disjunção com uma condição sempre verdadeira... apesar do usermail introduzido ser "vazio", a condição "vazio" OR True vai dar True, possibilitando assim a entrada no sistema.

Tentativa 4

Desta vez, o indivíduo voltou a entrar na conta de administrador, mas sem recorrer a SQL Injection – ele recorreu a um email válido e conhecido (este está exposto numa das páginas do sistema) e a uma palavra-passe “típica” e “básica”, “abc”; assim, tudo indica que na tentativa anterior, e após ter entrado na conta e ser redirecionado para uma página onde pode criar novos posts e mudar as credenciais de acesso (nomeadamente a password – ver GET no whireshark), este tenha mudado a palavra-passe.

Tentativa 5

Bastante idêntica à tentativa anterior, diria-se que ele próprio voltou a mudar a palavra-passe para ser vazia, isto porquê:

- se fosse apenas ele recorrer a SQL injection, ao colocar uma pelica significava que ele ia estar à procura de uma conta com palavra-passe "vazia", o que é deveras impossível em condições ditas normais existir na base de dados uma conta com palavra-passe vazia;

- ele também não iria conseguir aceder à conta ao colocar na palavra-passe apenas uma pelica ' e colocando o usermail real, porque no ficheiro de login.php, a query do usermail e password é um AND, ou seja, ambos os literais têm de ser verdadeiros; ia dar verdadeiro que ele encontrou na base de dados um utilizador com aquele e-mail (até porque se ele não tivesse encontrado, ia dar erro de DB Error), mas como não existe uma palavra-passe vazia, ou melhor, a palavra-passe daquele usermail não era vazia, iria dar certamente "erro de password inválida" ou coisa do género.

Assim, deduz-se que ele voltou a mudar a palavra-passe, mas desta vez para “vazio”, evidenciando aqui outra vulnerabilidade – falta de verificações no input para a mudança da credencial de palavra-passe (neste caso, teria de verificar no mínimo se o conteúdo do input seria diferente de “vazio”).

Restantes tentativas

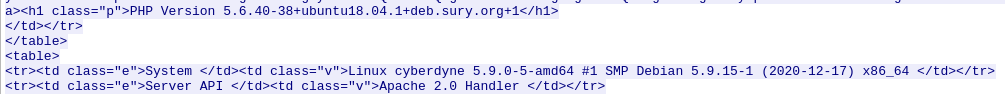
Nas restantes tentativas relativas à autenticação do atacante, ele recorreu sempre a um usermail real e no campo da palavra-passe voltou a recorrer a SQL Injection. Como já foi referido, após verificar que o usermail existe na base de dados, o query executado passa a ser:

$sql = "SELECT session FROM tblMembers WHERE username='" . $\_POST['usermail'] . "' AND password='" . $\_POST['password'] . "';";

Como é um AND, ambas as condições têm de ser verdadeiras; já sabemos que existe um usermail igual ao que foi introduzido, até porque ele é público e relativamente ao campo destinado à palavra-passe, ao colocar pelica ' ele está a "fechar" o campo da procura pela palavra-passe; mas como logo a seguir ele introduz uma condição verdadeira, a secção do SELECT destinada à password também vai dar verdadeira, possibilitando um reorno favorável da query e, assim, permitindo ao atacante entrar (de novo) na conta de administrador do sistema.

1. Acesso a páginas privadas

Enquanto navegava no sistema, o indivíduo conseguiu entrar na página info.php, páginas essa que não deveria estar disponível para este e que mostra mais informações acerca da versão PHP utilizada, do sistema operativo onde se encontra a VM, entre outras configurações:



1. Parâmetro type de GET products.php vulnerável a SQL Injection

Primeiramente, o atacante colocou no parâmetro do type:



Ao verificar o output do GET no whireshark, e como não houve nenhum erro, pode-se deduzir que o atacante acertou no número de atributos de uma tabela da base de dados.

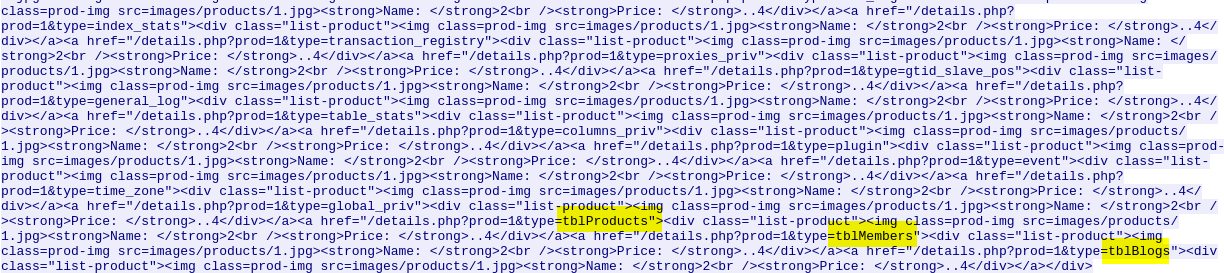
A seguir, ele tentou obter o nome da tabela da base de dados, mas sem sucesso porque, ao analisar-se o ficheiro products.php, é ele que está a gerar o html dinamicamente tendo em conta os resultados obtidos; pode-se verificar que ele apenas está a adquirir apenas os primeiros dois atributos do UNION SELECT:



Assim, ao mover-se o atributo TABLE\_NAME talvez seja possível obter algum resultado credível.

Desta vez, e tendo em conta o pensamento descrito anteriormente, ele testou o seguinte:





Como se pode observar pelo resultado obtido, o atacante conseguiu obter o nome das tabelas existentes na base de dados.

1. Parâmetro prod de GET details.php vulnerável a SQL Injection

Para começar, o indivíduo fez um UNION SELECT 1,2,3,4,5 e, pelo que se pode analisar no whireshark (pela falta de feedback/erros recebidos), ele deduz que existe alguma outra tabela igualmente com 5 atributos.

Depois, ele realizou um ,

Isto é, tentou escrever no servidor um ficheiro malicioso, ao qual obteve como resposta:

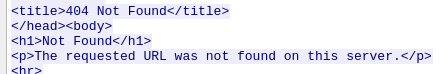


Efetuando alguma pesquisa na internet, soube-se que é impossível dar override a um ficheiro já existente recorrendo ao comando INTO OUTFILE (fonte: <https://stackoverflow.com/questions/960627/mysql-into-outfile-override-existing-file/960681>), pelo que este pode ser considerado um dos mecanismos de proteção utilizados.

A seguir, ele testou novamente criar este ficheiro malicioso, mas desta vez num diretório diferente, o /var/www/html, e obteve como resultado:

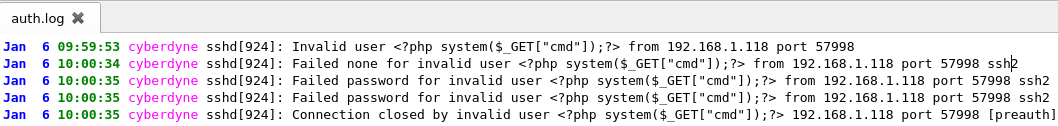


Ora, isto significa que em todas as tentativas que ele tentou criar um ficheiro x.txt malicioso, isto não foi possível, o que mostra uma das ocasiões em que o sistema barrou as más intenções do atacante. Ainda podemos verificar o resultado deste GET do ficheiro x.txt neste último diretório especificado, que reforça a ideia de que não foi possível criar ou modificar este mesmo ficheiro malicioso x.txt:



1. Parâmetro lang de GET display.php e recurso a web Shell e ficheiro auth.log

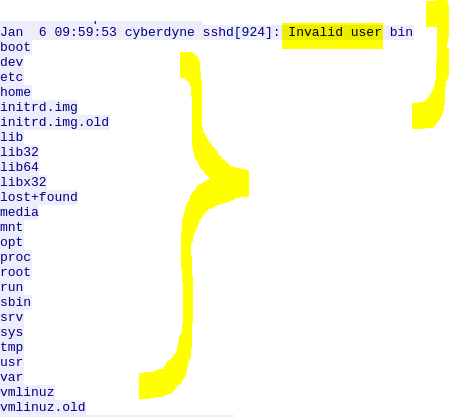
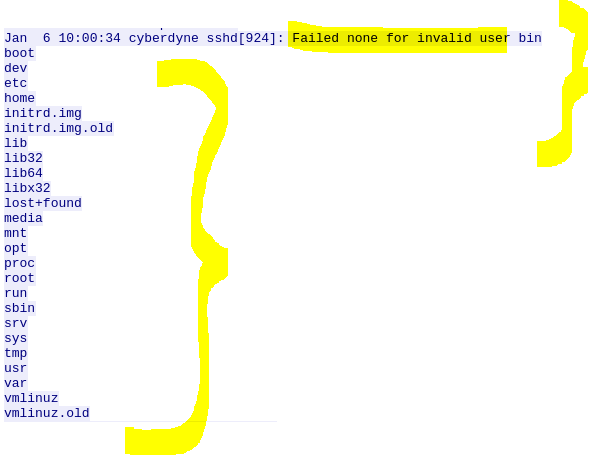
O indivíduo tentou conectar-se ao servidor através de ssh com um user igual a código php que representa uma web shell, isto é, uma shell normalmente maliciosa que permite executar comandos.

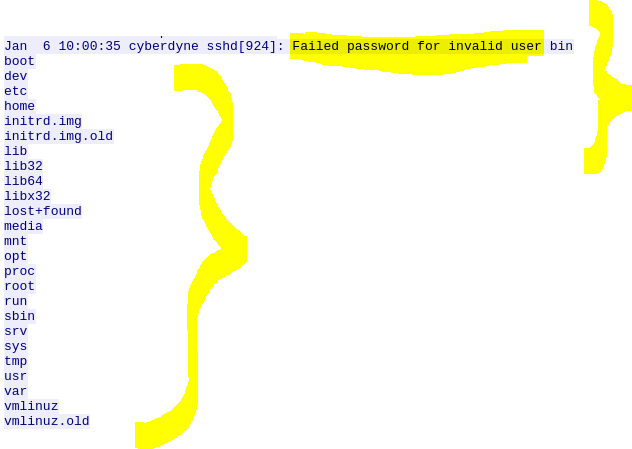
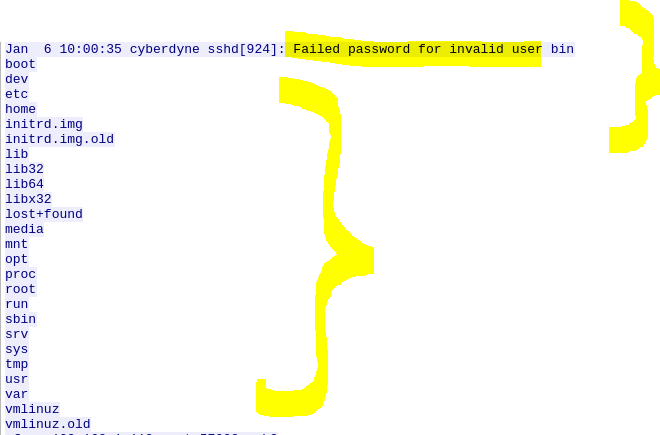


Como seria de esperar, o atacante não conseguiu conectar-se por ssh, mas este screenshot dos logs de autenticação será importante para o passo seguinte.

Agora, o criminoso faz um novo GET:

, no qual ele coloca no parâmetro “lang” o (conteúdo do) ficheiro de logs de autenticação. Ora, ao carregar esse mesmo ficheiro, e sabendo da existência do código php que lá se encontra, nomeadamente o campo que faz/está destinado a ler um parâmetro “cmd”, o atacante pode então acrescentar outro parâmetro no seu GET no display.php, o “cmd”; note-se que o comando passado é o ls, isto é, aquele que lista todos os diretórios no diretório atual:

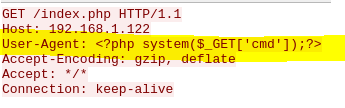
 



Ora, como se pode verificar neste conjunto de screesnhots, o atacante obteve como resposta todas as “frases” que existem no ficheiro auth.log (inclusive a repetição da linha associada à password), mas é notável que em vez de aparecer o código PHP, aparece o resultado do comando introduzido!

1. Parâmetro lang de GET display.php e recurso a web Shell e ficheiro access.log

Desta vez, o atacante realizou um GET do index.php, mas note-se um pequeno pormenor verificado no whireshark (ficheiro da captação da firewall) e no ficheiro access.log:

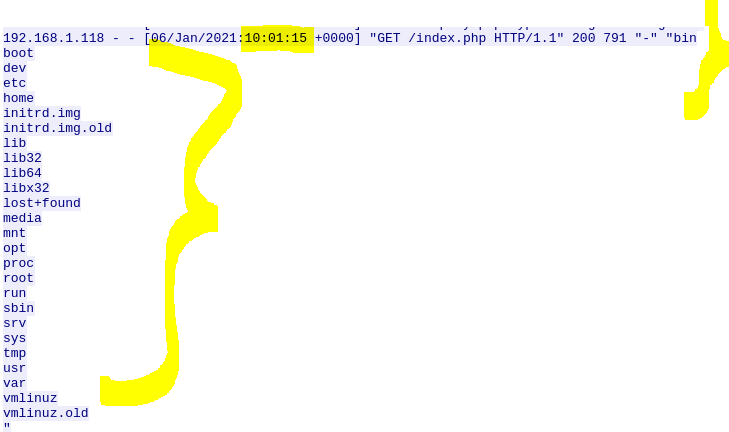




O atacante conseguiu modificar o user-agent!

 Assim, e de modo análogo ao tópico 5, ele irá conseguir executar comandos e obter os seus outputs! É só analisar:

No campo destinado ao parâmetro “lang”, ele carregou o ficheiro de log de acessos que, como sabemos, tem neste momento como última linha a mostrada anteriormente, isto é, um GET do index.php com um user-agent igual a uma web shell; Ora, como existe um campo que irá ler o GET do “cmd”, ele pode então dar da mesma forma um “cmd” e, assim, sempre que for verificar os resultados, irá sempre aparecer a mesma linha do ficheiro access.log, mas no lugar do user-agent, aparece o resultado do comando colocado como parâmetro e posteriormente executado (caso do comando ls, já se viu estes resultados anteriormente!):



Vulnerabilidades barradas

Como já foi dito anteriormente, o indivíduo tentou criar um ficheiro x.txt no servidor, mas este privilégio foi-lhe negado por diversas vezes; assim, este foi um dos processos barrados pelo sistema.

Posteriormente, o indivíduo tentou listar os containers de docker que estivessem a correr naquele determinado momento, mas como podemos verificar, ele não obteve nenhum resultado:





Ora, ao examinar este output, ou não existia nenhum container (o que é bastante improvável, tendo em conta a virtualização através de container de Docker ser bastante utilizada para colocar servidores online hoje em dia), ou o utilizador não teve permissões para tal. Observando o resultado de: , podemos verificar que o utilizador é de nome “www-data” – possivelmente um utilizador do sistema por default com poucas permissões de acesso, por isso este pode ser igualmente um mecanismo de confinamento.

Sequência de ações do atacante

Em primeiro lugar, o indivíduo tentou entrar no sistema recorrendo a SQL Injection, e após conseguir, modificou a palavra-passe por duas vezes.

Depois, andou a navegar entre páginas, entre elas a products.php, blog.php e info.php, para finalmente tentar descarregar um ficheiro de nome brochure.php, o que não foi possível:

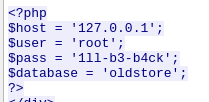


Pelo erro analisado no ficheiro error.log, deduz-se que este ficheiro não exista, pura e simplesmente; também não parece ter sido criado a partir do download.php, porque o ficheiro download.php importa o ficheiro getfile.php, e este é responsável por abrir ficheiros (em modo de leitura – “r”); deu erro no log de erros que fclose() recebeu como parâmetro um booleano 1 em vez de uma resource, ora, isto só confirma aquilo que já sabíamos, ele tentou abrir o ficheiro com o fopen() e como não conseguiu, a variável de retorno dessa função deu 1 (houve um problema com a função), pelo que fechar esse ficheiro neste caso é "fechar um valor booleano que simboliza que houve um erro".



Depois, ele tentou obter informações relativas à base de dados que sustém o sistema e escrever um ficheiro malicioso com recurso a SQL Injection.

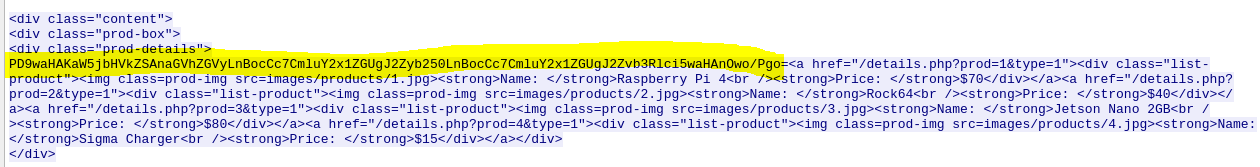
De seguida, voltou a tentar descarregar ficheiros provenientes do sistema, à qual a resposta foi favorável para a grande maioria das tentativas, possibilitando a obtenção de um Brochure.pdf e index.php, entre outros ficheiros, como o config.php, fornecendo informações confidenciais sobre o acesso à base de dados e, inclusive e finalmente, o nome da base de dados utilizada e que sustenta o sistema/servidor: “oldstore”:



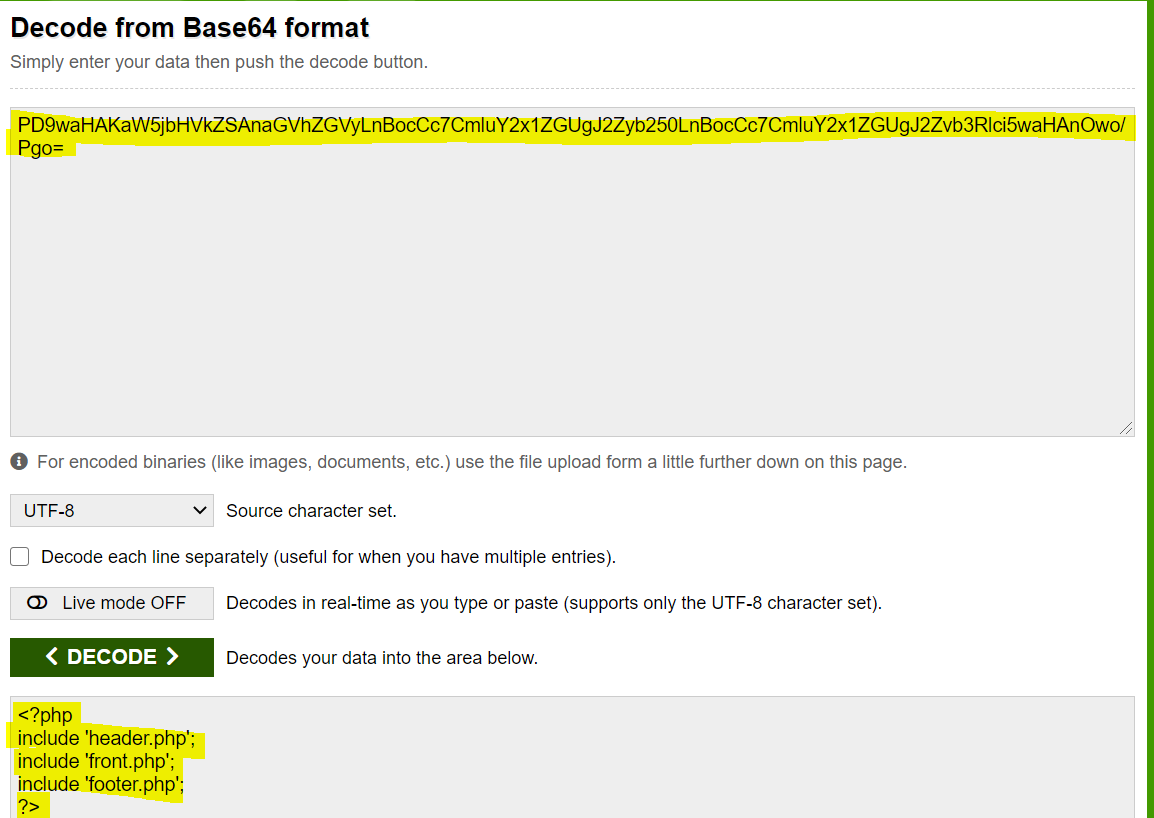
Outro dos ficheiros encontrados foi o display.php, que mostra dois parâmetros aos quais se pode efetuar um GET, um type (já anteriormente explorado) e um lang, que poderá dar mais informações ao atacante de como irá proceder a seguir.

 A seguir, ele efetua o GET seguinte:

- repare que um parâmetro colocado foi o de resource=index.php, ao qual obteve a seguinte resposta:



Tendo em conta o resultado suspeito, tentou-se analisar este conteúdo e meteu-se a hipótese de estar encriptado. Uma vez que o professor da unidade curricular falou numa aula para se suspeitar que algo codificado que acaba em “=” pudesse estar em base64, recorreu-se ao seguinte descodificador online: <https://www.base64decode.org/>



Pode-se concluir que o resultado codificado foi o do index.php!

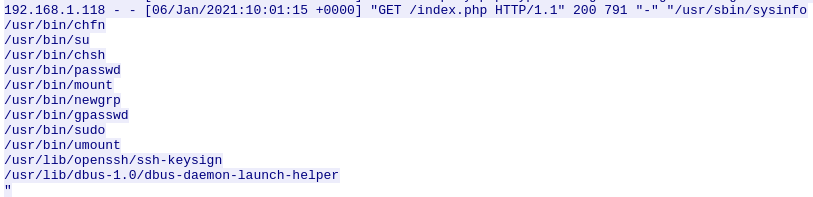
Depois, ele conseguiu executar comandos na máquina onde se encontra o servidor/sistema, recorrendo a código PHP presente nos logs de autenticação e de acesso; assim, conseguindo assim, por exemplo, informações sobre o sistema, recorrendo ao comando uname:





Outro dos comandos que ele executou foi o mount e o find / -mount que mostram a lista de ficheiros de sistema que foram montados.

Depois, ele executou o comando find / -perm -4000, ou seja, procurou quais os ficheiros e objetos com permissão de superutilizador!



Além disso, também obteve o conteúdo do ficheiro sysinfo, permitindo-se informar sobre configurações do sistema da máquina onde se encontra o servidor:

