code-breaking picklecode中对signed_cookies引擎分析

peri0d / 2019-09-12 08:39:24 / 浏览数 2186 安全技术 CTF 顶(0) 踩(0)

最近做了 ph 牛的 code-breaking，在做 picklecode 这一题时，没有搞懂那个 django 的 signed_cookies 引擎对 session 的操作，就 debug 了一下，做个总结，算是做了个代码审计吧

0x01 获取 session_auth_hash

题目：https://github.com/phith0n/code-breaking/tree/master/2018/picklecode

django 使用的 SESSION_ENGINE 为 `django.contrib.sessions.backends.signed_cookies`

pycharm 开启 debug 模式，username 为 peri0d，password 为 123456

入口文件在 `views.py`，第 34 行新建了用户并对密码进行了加密。第 35 行调用 `auth_login()` 函数，跳转到 `auth\__init__.py` 的 `login()` 方法

```
18  class RegistrationLoginView(LoginView):
19      def post(self, request, *args, **kwargs):  self: <challenge.views.RegistrationLoginView object at 0x00000282BC93BFD0>  request: <WSGIRequest: POST '/login/?next=/'
20          """
21          Handle POST requests: instantiate a form instance with the passed
22          POST variables and then check if it's valid.
23          """
24          form = self.get_form()  form: <AuthenticationForm bound=True, valid=False, fields=(username;password)>
25          if form.is_valid():
26              return self.form_valid(form)
27
28          if 'username' not in form.cleaned_data or 'password' not in form.cleaned_data:
29              return self.form_invalid(form)
30
31          if User.objects.filter(username=form.cleaned_data['username']).exists():
32              return self.form_invalid(form)
33
34          user = User.objects.create_user(form.cleaned_data['username'], None, form.cleaned_data['password'])  user: peri0d
35          auth_login(request, user)
36          return HttpResponseRedirect(self.get_success_url())
37
```

第 97 行，调用 user 类的 `get_session_auth_hash()` 方法来获取 `session_auth_hash` 的值，跟进 `get_session_auth_hash()`

```
87  def login(request, user, backend=None):  request: <WSGIRequest: POST '/login/?next=/'>  user: peri0d  backend: None
88      """
89      Persist a user id and a backend in the request. This way a user doesn't
90      have to reauthenticate on every request. Note that data set during
91      the anonymous session is retained when the user logs in.
92      """
93      session_auth_hash = ''  session_auth_hash: ''
94      if user is None:
95          user = request.user
96      if hasattr(user, 'get_session_auth_hash'):
97          session_auth_hash = user.get_session_auth_hash()
98
```

给 `key_salt` 赋值后调用 `salted_hmac(key_salt, self.password)` 生成 `session_auth_hash`，这里的 `password` 是经过加密的，跟进 `salted_hmac()`

```
123     def get_session_auth_hash(self):  self: peri0d
124         """
125         Return an HMAC of the password field.
126         """
127         key_salt = "django.contrib.auth.models.AbstractBaseUser.get_session_auth_hash"  key_salt: 'django.contrib.auth.models.AbstractBaseUser.get_session_auth_hash'
128         return salted_hmac(key_salt, self.password).hexdigest()
```

在第 39 行对 `key_salt + secret` 进行 sha1 加密并以 byte 类型返回给 `key`。这里的 `value` 是经过加密后的 `password`。然后调用 `hmac.new()`返回一个 sha1 模式的 `hmac` 对象

```
def salted_hmac(key_salt, value, secret=None):  key_salt: b'django.contrib.auth.models.AbstractBaseUser.get_session_auth_hash'
    """
    Return the HMAC-SHA1 of 'value', using a key generated from key_salt and a
    secret (which defaults to settings.SECRET_KEY).

    A different key_salt should be passed in for every application of HMAC.
    """
    if secret is None:
        secret = settings.SECRET_KEY

    key_salt = force_bytes(key_salt)
    secret = force_bytes(secret)

    # We need to generate a derived key from our base key.  We can do this by
    # passing the key_salt and our base key through a pseudo-random function and
    # SHA1 works nicely.
    key = hashlib.sha1(key_salt + secret).digest()  key: b'\x03(A\x8a\xaa=sP\xd8\x90\x00F\xce\xca\xcb6\xb3H\x1f\xde'

    # If len(key_salt + secret) > sha_constructor().block_size, the above
    # line is redundant and could be replaced by key = key_salt + secret, since
    # the hmac module does the same thing for keys longer than the block size.
    # However, we need to ensure that we *always* do this.
    return hmac.new(key, msg=force_bytes(value), digestmod=hashlib.sha1)
```


流程梳理

```
key_salt = '***'
# SECRET_KEY
secret = '*******'
key = hashlib.sha1(key_salt + secret).digest()
sha1_obj = hmac.new(key, msg=password_enc, digestmod=hashlib.sha1)
session_auth_hash = sha1_obj.hexdigest()
```

0x02 初始化 sessionid

获取 session_auth_hash 后，单步调试，进入 base.py 执行 __contains__() 函数，参数为 _auth_user_id

```
def __contains__(self, key):  self: <django.contrib.sessions.backends.signed_cookies.SessionStore object at 0x0000026F8C894E80>  key: '_auth_user_id'
    return key in self._session
```


单步调试，然后执行 _get_session() 函数，返回缓存 session，是一个空字典

```python
def _get_session(self, no_load=False):
    """
    Lazily load session from storage (unless "no_load" is True, when only
    an empty dict is stored) and store it in the current instance.
    """
    self.accessed = True
    try:
        return self._session_cache
    except AttributeError:
        if self.session_key is None or no_load:
            self._session_cache = {}
        else:
            self._session_cache = self.load()
    return self._session_cache

_session = property(_get_session)
```


在第 108 行执行 `cycle_key()`，使用新密钥保存相同的数据，调用 `save()`，它在请求结束时自动保存一个带有新密钥的 cookie 。

```python
99      if SESSION_KEY in request.session:
100         if _get_user_session_key(request) != user.pk or (
101                 session_auth_hash and
102                 not constant_time_compare(request.session.get(HASH_SESSION_KEY, ''), session_auth_hash)):
103             # To avoid reusing another user's session, create a new, empty
104             # session if the existing session corresponds to a different
105             # authenticated user.
106             request.session.flush()
107     else:
108         request.session.cycle_key()
109
```



```python
def cycle_key(self):  self: <django.contrib.sessions.backends.signed_cookies.SessionStore object at 0x000002828C5AF048>
    """
    Keep the same data but with a new key. Call save() and it will
    automatically save a cookie with a new key at the end of the request.
    """
    self.save()
```


跟进 `save()`，在第 41 行执行 `_get_session_key()` ，生成一个 base64 编码后的字符串作为 `session key`，继续跟进，它又调用了 `signing.dumps()`

```python
35      def save(self, must_create=False):
36          """
37          To save, get the session key as a securely signed string and then set
38          the modified flag so that the cookie is set on the client for the
39          current request.
40          """
41          self._session_key = self._get_session_key()
42          self.modified = True
```

```
def _get_session_key(self):  self: <django.contrib.sessions.backends.signed_cookies.SessionStore object at 0x000002828C5AF048>
    """
    Instead of generating a random string, generate a secure url-safe
    base64-encoded string of data as our session key.
    """
    return signing.dumps(
        self._session, compress=True,
        salt='django.contrib.sessions.backends.signed_cookies',
        serializer=self.serializer,
    )
```


然后单步调试进入到 `_get_session()` 方法获取 `self._session`，从缓存中加载 session，此时为一个空字典，即 `self._session == {}`

```
def _get_session(self, no_load=False):
    """
    Lazily load session from storage (unless "no_load" is True, when only
    an empty dict is stored) and store it in the current instance.
    """
    self.accessed = True
    try:
        return self._session_cache
    except AttributeError:
        if self.session_key is None or no_load:
            self._session_cache = {}
        else:
            self._session_cache = self.load()
    return self._session_cache
```


然后分别给 compress，salt，serializer 赋值，然后调用 `signing.dumps()` ，继续跟进，传入的参数 `obj = {}, salt = 'django.contrib.sessions.backends.signed_cookies', compress = True`

```python
def dumps(obj, key=None, salt='django.core.signing', serializer=JSONSerializer, compress=False):
    """
    Return URL-safe, hmac/SHA1 signed base64 compressed JSON string. If key is
    None, use settings.SECRET_KEY instead.

    If compress is True (not the default), check if compressing using zlib can
    save some space. Prepend a '.' to signify compression. This is included
    in the signature, to protect against zip bombs.

    Salt can be used to namespace the hash, so that a signed string is
    only valid for a given namespace. Leaving this at the default
    value or re-using a salt value across different parts of your
    application without good cause is a security risk.

    The serializer is expected to return a bytestring.
    """
    data = serializer().dumps(obj)  data: b'\x80\x03}q\x00.'

    # Flag for if it's been compressed or not
    is_compressed = False  is_compressed: False

    if compress:
        # Avoid zlib dependency unless compress is being used
        compressed = zlib.compress(data)  compressed: b'x\x9c`\xae-d\xd0\x03\x00\x06\x8a\x01\xa0'
        if len(compressed) < (len(data) - 1):
            data = compressed
            is_compressed = True
    base64d = b64_encode(data).decode()  base64d: 'gAN9cQAu'
    if is_compressed:
        base64d = '.' + base64d
    return TimestampSigner(key, salt=salt).sign(base64d)
```


在 `signing.dumps()` 中对序列化之后的数据进行压缩，然后进行 base64 编码，再 `decode()` 为一个 Unicode 的 `base64d`，其值为 `'gAN9cQAu'`，最后调用 `TimestampSigner` 类的 `sign()` 方法，继续跟进

`TimestampSigner` 类继承自 `Signer` 类，先调用它的 `__init__` 方法进行初始化，`key = 'zs%o-mvuihtk6g4pgd+xpa&1hh9%&ulnf!@9qx8_y5kk+7^cvm', sep = ':', salt = 'django.contrib.sessions.backends.signed_cookies'`

```
class Signer:

    def __init__(self, key=None, sep=':', salt=None):
        # Use of native strings in all versions of Python
        self.key = key or settings.SECRET_KEY
        self.sep = sep
        if _SEP_UNSAFE.match(self.sep):
            raise ValueError(
                'Unsafe Signer separator: %r (cannot be empty or consist of '
                'only A-z0-9-_=)' % sep,
            )
        self.salt = salt or '%s.%s' % (self.__class__.__module__, self.__class__.__name__)
```


然后调用 TimestampSigner 类的 sign() 方法，根据 value='gAN9cQAu'，sep 和 timestamp() 对 value 进行重新赋值，其值为 'gAN9cQAu:1i5q6e' ，然后再次在 Signer.sign() 中重新赋值，得到最后结果 'gAN9cQAu:1i5q6e:wjJR2MUONx_wmPA3m8zYqTj5uCQ

```
class TimestampSigner(Signer):

    def timestamp(self):
        return baseconv.base62.encode(int(time.time()))

    def sign(self, value):  self: <django.core.signing.TimestampSigner object at 0x000002828C07E668>  value: 'gAN9cQAu:1i5nte'
        value = '%s%s%s' % (value, self.sep, self.timestamp())
        return super().sign(value)
```



```
def signature(self, value):
    return base64_hmac(self.salt + 'signer', value, self.key)

def sign(self, value):
    return '%s%s%s' % (value, self.sep, self.signature(value))
```


回到 save() ，继续单步调试，调用了 base.py 中第 170 行的 _set_session_key() 方法，将 value 赋值给 session_key 和 _session_key

```
160     def _validate_session_key(self, key):
161         """
162         Key must be truthy and at least 8 characters long. 8 characters is an
163         arbitrary lower bound for some minimal key security.
164         """
165         return key and len(key) >= 8
166
167     def _get_session_key(self):
168         return self.__session_key
169
170     def _set_session_key(self, value):
171         """
172         Validate session key on assignment. Invalid values will set to None.
173         """
174         if self._validate_session_key(value):
175             self.__session_key = value
176         else:
177             self.__session_key = None
178
179     session_key = property(_get_session_key)
180     _session_key = property(_get_session_key, _set_session_key)
```


回到 `save()` ，完成赋值，回到 `cycle_key()`，再回到 `auth\__init__.py` 的 `login()` 方法的第 108 行，这时可以在变量列表看到设置的 session 信息了

```
▶ _SessionBase__not_given = {object} <object object at 0x0000026F8C134160>
  _SessionBase__session_key = {str} 'gAN9cQAu:1i5q6e:wjJR2MUONx_wmPA3m8zYqTj5uCQ'
▶ _session = {dict} <class 'dict'>: {'_auth_user_id': '1', '_auth_user_backend': 'django.contrib.auth.backends.ModelBackend', '_auth_user_hash': 'd3d0ad9c10fe19243554be9d65090394af66af6b'}
▶ _session_cache = {dict} <class 'dict'>: {'_auth_user_id': '1', '_auth_user_backend': 'django.contrib.auth.backends.ModelBackend', '_auth_user_hash': 'd3d0ad9c10fe19243554be9d65090394af66af6b'}
  _session_key = {str} 'gAN9cQAu:1i5q6e:wjJR2MUONx_wmPA3m8zYqTj5uCQ'
  accessed = {bool} True
  modified = {bool} True
▶ serializer = {type} <class 'core.serializer.PickleSerializer'>
  session_key = {str} 'gAN9cQAu:1i5q6e:wjJR2MUONx_wmPA3m8zYqTj5uCQ'
```


后面的代码是 django 对用户的持久化处理以及对 CSRF token 的验证等等，值得注意的是在第 126 行到 128 行，进行了 session 设置

```
110         try:
111             backend = backend or user.backend
112         except AttributeError:
113             backends = _get_backends(return_tuples=True)  backends: <class 'list'>: [(<django.contrib.auth.backends.Model
114             if len(backends) == 1:
115                 _, backend = backends[0]  _: <django.contrib.auth.backends.ModelBackend object at 0x0000026F8EA17358>
116             else:
117                 raise ValueError(
118                     'You have multiple authentication backends configured and '
119                     'therefore must provide the `backend` argument or set the '
120                     '`backend` attribute on the user.'
121                 )
122         else:
123             if not isinstance(backend, str):
124                 raise TypeError('backend must be a dotted import path string (got %r).' % backend)
125 
126         request.session[SESSION_KEY] = user._meta.pk.value_to_string(user)
127         request.session[BACKEND_SESSION_KEY] = backend
128         request.session[HASH_SESSION_KEY] = session_auth_hash
129         if hasattr(request, 'user'):
130             request.user = user
131         rotate_token(request)
132         user_logged_in.send(sender=user.__class__, request=request, user=user)
133 
```



```
_session = {dict} <class 'dict'>: {'_auth_user_id': '1', '_auth_user_backend': 'django.contrib.auth.back
 '_auth_user_id' (2322540718448) = {str} '1'
 '_auth_user_backend' (2322540736848) = {str} 'django.contrib.auth.backends.ModelBackend'
 '_auth_user_hash' (2322540718512) = {str} 'c14f64e1fa9a625a4a661487eb858d1faee050b0'
 __len__ = {int} 3
```


流程梳理

```
_session = {}
# SECRET_KEY
secret = '******'
salt='****'

data = serializer().dumps(_session)
compressed = zlib.compress(data)
base64d = b64_encode(data).decode()

session_key = TimestampSigner(SECRET_KEY, salt=salt).sign(base64d)
```

0x03 response 写入 session

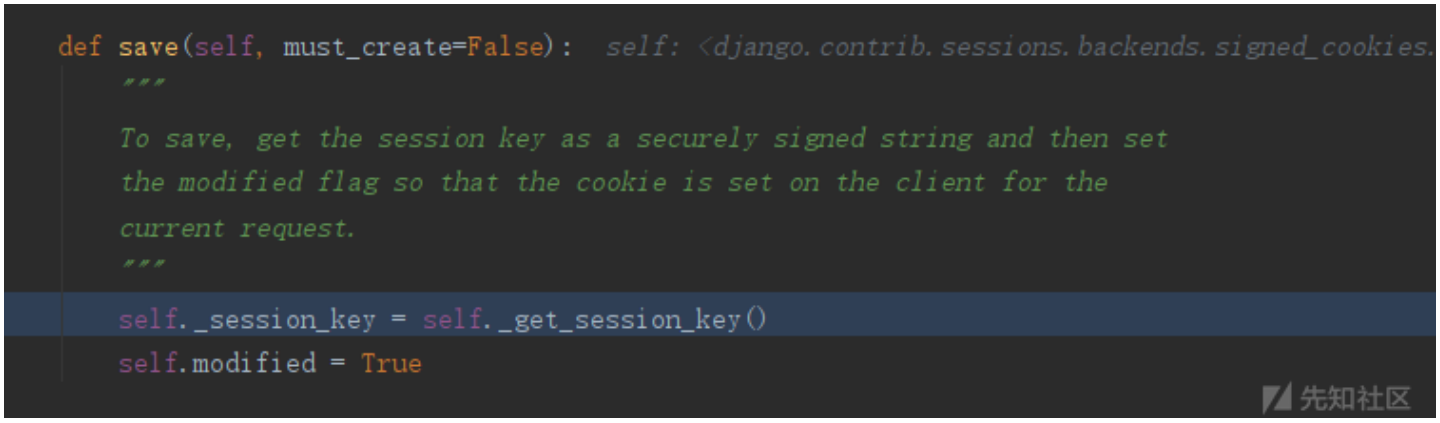
然后看它如何在 response 中设置 cookie 的，继续调试，在 `contrib\sessions\middleware.py` 中发现其对 cookie 的操作，从 44 行开始是设置 cookie 的存活时间，在第 58 行看到了 `save()` 函数，进行 cookie 的保存，单步调试进入

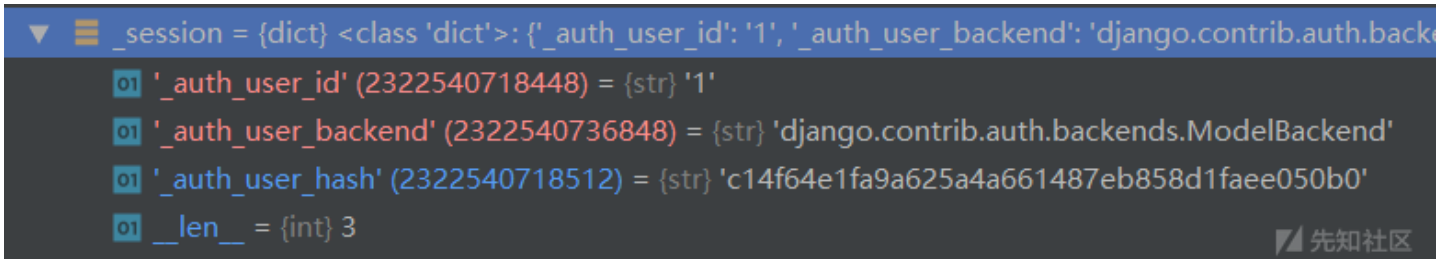
```
22      def process_response(self, request, response):  self: <django.contrib.sessions.middleware.Session
23          """
24          If request.session was modified, or if the configuration is to save the
25          session every time, save the changes and set a session cookie or delete
26          the session cookie if the session has been emptied.
27          """
28          try:
29              accessed = request.session.accessed  accessed: True
30              modified = request.session.modified  modified: True
31              empty = request.session.is_empty()  empty: False
32          except AttributeError:
33              pass
34          else:
35              # First check if we need to delete this cookie.
36              # The session should be deleted only if the session is entirely empty
37              if settings.SESSION_COOKIE_NAME in request.COOKIES and empty:
38                  response.delete_cookie(
39                      settings.SESSION_COOKIE_NAME,
40                      path=settings.SESSION_COOKIE_PATH,
41                      domain=settings.SESSION_COOKIE_DOMAIN,
42                  )
43              else:
44                  if accessed:
45                      patch_vary_headers(response, ('Cookie',))
46                  if (modified or settings.SESSION_SAVE_EVERY_REQUEST) and not empty:
47                      if request.session.get_expire_at_browser_close():
48                          max_age = None  max_age: 1209600
49                          expires = None  expires: 'Thu, 19 Sep 2019 13:02:23 GMT'
```



```
50                      else:
51                          max_age = request.session.get_expiry_age()
52                          expires_time = time.time() + max_age  expires_time: 1568898143.5757177
53                          expires = http_date(expires_time)
54                      # Save the session data and refresh the client cookie.
55                      # Skip session save for 500 responses, refs #3881.
56                      if response.status_code != 500:
57                          try:
58                              request.session.save()
59                          except UpdateError:
60                              raise SuspiciousOperation(
61                                  "The request's session was deleted before the "
62                                  "request completed. The user may have logged "
63                                  "out in a concurrent request, for example."
64                              )
65                          response.set_cookie(
66                              settings.SESSION_COOKIE_NAME,
67                              request.session.session_key, max_age=max_age,
68                              expires=expires, domain=settings.SESSION_COOKIE_DOMAIN,
69                              path=settings.SESSION_COOKIE_PATH,
70                              secure=settings.SESSION_COOKIE_SECURE or None,
71                              httponly=settings.SESSION_COOKIE_HTTPONLY or None,
72                              samesite=settings.SESSION_COOKIE_SAMESITE,
73                          )
74          return response
```

在 save() 函数中，调用 _get_session_key() 函数，剩下的反序列化和前面的相同，只是 session 的值发生了改变，从空字典变为含有 3 个元素的字典，然后就是将 cookie 设置在返回包中，这就完成了 cookie 设置的分析





0x04 总结

总结一下，它对 session 处理的核心机制在于 `django.core.signing.dumps()` 函数，其具体代码如下，可以看到，`data` 为 pickle 序列化之后的 byte 对象，我们只要将 `data` 改为构造好的 evil pickle code 即能实现任意的代码执行

```
def dumps(obj, key=None, salt='django.core.signing', serializer=JSONSerializer, compress=False):

    data = serializer().dumps(obj)

    is_compressed = False

    if compress:
        compressed = zlib.compress(data)
        if len(compressed) < (len(data) - 1):
            data = compressed
            is_compressed = True
    base64d = b64_encode(data).decode()
    if is_compressed:
        base64d = '.' + base64d
    return TimestampSigner(key, salt=salt).sign(base64d)
```


上一篇：通过修改源代码达到菜刀无特征链接 下一篇：ByteCTF 2019 Writ...

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板