AGENDA

- Introduction
- Why
- What
- How
@WIREGHOUL

- Husband
- Father
- Penetration tester
- Geek
- Blogger – http://www.justanotherhacker.com
- Projects
  - htshells
  - Graudit
  - Doona and more
- Contributor
  - Nikto
  - Dotdotpwn
  - PadBuster and more
INTRODUCTION – PAYMENT GATEWAY

A payment gateway is an e-commerce application service provider service that authorizes payments online. It is the equivalent of a physical point of sale terminal. Payment gateways protect credit card details by encrypting sensitive information, such as credit card numbers, to ensure that information is passed securely between the customer and the merchant and also between merchant and the payment processor.
INTRODUCTION

- Actors
- Definitions
- Payment gateway APIs
- Design vulnerabilities
- Cryptography
- Implementation bugs
MERCHANT
PAYMENT GATEWAY
ATTACKER
TESTING PAYMENT

• Use test card numbers

• VISA 4111 1111 1111 1111

• Mastercard 5555 5555 5555 4444

• American Express 378282246310005
API

• Primary means of interaction between online payment form and payment gateway
• Typical operations include:
  • Charge card
  • Query payment status
  • Manage recurring payments
  • Refund payments
API ACCESS POINTS

• Production
  • https://api.paymentgateway.url

• Sandbox
  • https://test.paymentgateway.url
LEVERAGING THE SANDBOX

Error 506

Invalid account configuration. Please contact the merchant.
API - DIRECT

POST /payment HTTP/1.1
Host: www.webshop.com
...
Creditcard=4111111111111111
ExpM=04
ExpY=15
CCV=123
Amount=100.00
API - DIRECT

POST /payment HTTP/1.1
Host: www.paymentgw.com
SOAP-Action: Payment

....
<CardNumber>4111111111111111</CardNumber>
<Expiry>
  <Month>04</Month>
  <Year>14</Year>
</Expiry>
<CVV>123</CVV>
<Amount>10000</Amount>
API - DIRECT

GET

HTTP/1.1 200 OK
Connection: close
....
<Response>
<Code>0</Code>
<Receipt>912937791-0008912</Receipt>
<Amount>10000</Amount>
....

Custom

Payment Gateway

XML
HTTP/1.1 200 OK
Connection: close

....

<H1>Payment received</H1>
Thank you for shopping at webshop.com your receipt number is:
<b>912937791-0008912</b>
...

Customer

GET

Payment Gateway

POST

XML

HTML

API - DIRECT
API – HOSTED DIRECT POST

GET /checkout HTTP/1.1
Host: www.webshop.com

...
API – HOSTED DIRECT POST

GET

HTTP/1.1 200 OK
Connection: close
....
<form action=https://paymentgw.com/svc/pay method=post>
<input name=Credita>
<input name=expMonth>
<input name=expYear>
<input name=CVV>
<input name=amount value=10000>
...

Custom

Payment Gateway
API – HOSTED DIRECT POST

POST /svc/pay HTTP/1.1
Host: paymentgw.com

... Creditcard=411111111111111
expMonth=04
expYear=15
CCV=123
amount=100.00
API – HOSTED DIRECT POST

HTTP/1.1 302 Moved
Location: https://www.webshop.com/return?response=sucess&amount=10000&receipt=912937791-0008912&transactionID=123....
API – HOSTED DIRECT POST

API – HOSTED DIRECT POST

HTTP/1.1 200 OK
Connection: close

....

<H1>Payment received</H1>
Thank you for shopping at webshop.com your receipt number is:
<b>912937791-0008912</b>

....
API – HOSTED REDIRECT

GET /checkout HTTP/1.1
Host: www.webshop.com
...

Customer

GET

Payment Gateway
API – HOSTED REDIRECT

HTTP/1.1 302 Moved
Location:
API – HOSTED REDIRECT

GET

302

GET

API – HOSTED REDIRECT

```
HTTP/1.1 200 OK
Connection: close

....
<form action=https://paymentgw.com/svc/pay method=post>
<input name=Creditcard>
<input name=expMonth>
<input name=expYear>
<input name=CVV>
<input name=amount value=10000>
<input name=returnurl value=https://www.webshop.com/return>
....
```
API – HOSTED REDIRECT

POST /svc/pay HTTP/1.1
Host: paymentgw.com

... Creditcard=4111111111111111
expMonth=04
expYear=15
CCV=123
amount=100.00
returnurl=https://www.webshop.com/return

GET

302
API – HOSTED REDIRECT

HTTP/1.1 302 Moved
Location:
https://www.webshop.com/return?response=succes&amount=10000&receipt=912937791-0008912&transactionID=123....
API – HOSTED REDIRECT

GET

302

GET

302 redirect

API – HOSTED REDIRECT

HTTP/1.1 200 OK
Connection: close
....
<H1>Payment received</H1>
Thank you for shopping at webshop.com your receipt number is:
<b>912937791-0008912</b>

....
API – HOSTED RE-DIRECT ALTERNATIVE

GET /checkout HTTP/1.1
Host: www.webshop.com

...
API – HOSTED RE-DIRECT ALTERNATIVE

HTTP/1.1 302 Moved
API – HOSTED RE-DIRECT ALTERNATIVE

GET

API – HOSTED RE-DIRECT ALTERNATIVE

HTTP/1.1 200 OK
Connection: close

<form action=https://paymentgw.com/svc/pay method=post>
  <input name=Creditcard>
  <input name=expMonth>
  <input name=expYear>
  <input name=CVV>
  <input name=amount value=10000>
  <input name=returnurl value=https://www.webshop.com/return>
</form>
API – HOSTED RE-DIRECT ALTERNATIVE

POST /svc/pay HTTP/1.1
Host: paymentgw.com

... 
Creditcard=4111111111111111
expMonth=04
expYear=15
CCV=123
amount=100.00
returnurl=https://www.webshop.com/return
API – HOSTED RE-DIRECT ALTERNATIVE

POST /return HTTP/1.1
Host: www.webshop.com

... response=succcess
amount=10000
receipt=912937791-0008912
transactionID=123
....
API – HOSTED RE-DIRECT ALTERNATIVE

GET

302

GET

HTTP/1.1 200 OK
Connection: close
…

POST

HTML

Payment Gateway
API – HOSTED RE-DIRECT ALTERNATIVE

HTTP/1.1 302 Moved
Location: https://www.webshop.com/thanks.html
API – HOSTED RE-DIRECT ALTERNATIVE

GET /thanks.html HTTP/1.1
Host: www.webshop.com

GET /thanks.html HTTP/1.1
Host: www.webshop.com

302 redirect

302

GET

POST

HTML

GET

Payment Gateway

Customer

Merchant

Payment Gateway
API – HOSTED RE-DIRECT ALTERNATIVE

Custom

GET

302

HTTP/1.1 200 OK
Connection: close
....
<H1>Payment received</H1>
Thank you for shopping at webshop.com
...

GET

302 redirect

HTML

Payment Gateway
API - DIRECT

POST /refundCard HTTP/1.1
Host: www.paymentgw.com

…

<Receipt>912937791-0008912</Receipt>
<Amount>10000</Amount>
<Hash>92deb9c1596dbc30003198184c31ef52</Hash>
HTTP/1.1 200 OK
Connection: close
....
<Response>
<Code>1</Code>
<Receipt>912937791-0008912/01</Receipt>
<Amount>10000</Amount>
....
TRADITIONAL ATTACKS
TRADITIONAL ATTACKS
TRADITIONAL ATTACKS

• Change payment amount
  https://paymentgateway.com/pay? amount=0.01
  Solved with request validation!

• Spoof payment received message to return url
  https://merchant.com/return?Success=1&Amount=100.00&Message=Paid
  Solved with response validation!
REQUEST VALIDATION

- To validate the request of the payment page result, signed request is often used - which is the result of the hash function in which the parameters of an application confirmed by a «secret word», known only to the merchant and payment gateway.
REQUEST VALIDATION

• Protects the “vital” details of the transaction

Example:

• SHA1 of MERCHANTID, TXNTYPE, REFERENCEID, AMOUNT, CURRENCY, TIMESTAMP
REQUEST VALIDATION EXAMPLE

sha1('ABC9999|password123|1|Invoice 986616|100.00|20140121222324')

4e65a02daacaf2f94f057fbc3d09c43883d10dc8

md5('password123abc9999100.00aud')

ce9b54a5bc2f08dd2a2bf5f3b2d2d8f0

md5(md5('20140121222324.ABC99999.Invoice 986616.100.AUD').'.Secrit123')

6a0a4eb970340d98fa33daf21400e5eb
RESPONSE VALIDATION

- Protects the “vital” details of the payment receipt

- Example:
  - SHA1 of MERCHANTID, TRANSACTIONID, AMOUNT
**RESPONSE VALIDATION EXAMPLE**

<table>
<thead>
<tr>
<th>SHA1 Calculation</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha1('ABC9999</td>
<td>Secrit123</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MD5 Calculation</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5('secrit123saltabc9999approved1-918490ae-9a1c-11de')</td>
<td>04beffd2eaf481e0d50ef2134188c6d0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MD5 of MD5 Calculation</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5(md5('20140121222324.ABC99999.Invoice 986616.00.Completed.auth.0000').'.Secrit123')</td>
<td>1f35ae73cf918f446cc45875948bd300</td>
</tr>
</tbody>
</table>
ABUSING REQUEST VALIDATION

- Bypass validation
- Abuse cryptographic properties
- Defeat secret key
BYPASSING REQUEST VALIDATION

- HTTP Parameter Pollution
  https://url/pay?amount=100.00&amount=0.01

- Abusing unprotected parameters
  https://url/pay?expiry_date=31/12/2099

- Abusing application logic
  https://url/pay?pre_auth=1
ABUSE CRYPTOGRAPHIC PROPERTIES

WHAT IF

THEY USED A LENGTH EXTENSION ATTACK?
LENGTH EXTENSION ATTACK

- The reason $H(k|m)$ is not the standard comes from the message extension attack
- Hashes operate on block of text
- Padding is used to fill out the blocks
- Attacker knows $H(k|m)$ and $m$
- Compute $H(k|m|p|m_2)$
- $p$ is the padding that would have applied to $k|m$
- $m_2$ is an arbitrary message
- Attacker can now use $H(k|m|p|m_2)$ and $m|p|m_2$ to pass validation checks
LENGTH EXTENSION ATTACKS ARE COSTLY
## DEFEATING REQUEST VALIDATION

<table>
<thead>
<tr>
<th>Fingerprint</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EPS_MERCHANTID</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td></td>
</tr>
<tr>
<td>EPS_TXNTYPE</td>
<td></td>
</tr>
<tr>
<td>EPS_REFERENCEID</td>
<td></td>
</tr>
<tr>
<td>EPS_AMOUNT</td>
<td></td>
</tr>
<tr>
<td>EPS_TIMESTAMP</td>
<td></td>
</tr>
</tbody>
</table>
DEFEATING REQUEST VALIDATION

<input hidden EPS_MERCHANT = “ABC999”>
<input hidden EPS_TXNTYPE = "0”>
<input hidden EPS_REFERENCEID ="Invoice 986616”>
<input hidden EPS_AMOUNT ="100.00”>
<input hidden EPS_TIMESTAMP ="20140121222324”>
<input hidden EPS_FINGERPRINT =“5f330cea9480efd63669b1b1464db1339c964bdf”>
<input hidden EPS_RESULTURL = “https://www.merchantsite.com/”>
## DEFEATING REQUEST VALIDATION

<table>
<thead>
<tr>
<th>Fingerprint</th>
<th>Web form</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPS_MERCHANTID</td>
<td>EPS_MERCHANT</td>
</tr>
<tr>
<td>Password</td>
<td></td>
</tr>
<tr>
<td>EPS_TXNTYPE</td>
<td>EPS_TXNTYPE</td>
</tr>
<tr>
<td>EPS_REFERENCEID</td>
<td>EPS_REFERENCEID</td>
</tr>
<tr>
<td>EPS_AMOUNT</td>
<td>EPS_AMOUNT</td>
</tr>
<tr>
<td>EPS_TIMESTAMP</td>
<td>EPS_TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>EPS_FINGERPRINT</td>
</tr>
</tbody>
</table>
### DEFEATING REQUEST VALIDATION

<table>
<thead>
<tr>
<th>Fingerprint</th>
<th>Web form</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPS_MERCHANTID</td>
<td>EPS_MERCHANT</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td></td>
</tr>
<tr>
<td>EPS_TXNTYPE</td>
<td>EPS_TXNTYPE</td>
</tr>
<tr>
<td>EPS_REFERENCEID</td>
<td>EPS_REFERENCEID</td>
</tr>
<tr>
<td>EPS_AMOUNT</td>
<td>EPS_AMOUNT</td>
</tr>
<tr>
<td>EPS_TIMESTAMP</td>
<td>EPS_TIMESTAMP</td>
</tr>
<tr>
<td>EPS_FINGERPRINT</td>
<td><strong>EPS_FINGERPRINT</strong></td>
</tr>
</tbody>
</table>
# DEFEATING REQUEST VALIDATION

<table>
<thead>
<tr>
<th>Fingerprint</th>
<th>Web form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC0010</td>
<td>ABC9999</td>
</tr>
<tr>
<td><strong>Secrit123</strong></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test reference</td>
<td>Invoice 986616</td>
</tr>
<tr>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>20120916221931</td>
<td>20140121222324</td>
</tr>
<tr>
<td></td>
<td><strong>5f330cea9480efd63669b1b1464db1339c964bdf</strong></td>
</tr>
</tbody>
</table>
SHARED SECRET

- Shared secret is usually:
- Vendor supplied
- Never changes
- Sometimes converted to upper/lower-case
<table>
<thead>
<tr>
<th>Characters</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-z0-9</td>
<td>8</td>
</tr>
<tr>
<td>a-zA-Z0-9</td>
<td>8</td>
</tr>
<tr>
<td>a-zA-Z0-9!@#$%^&amp;*()[]_+-=+;:&quot;&quot;,./?</td>
<td>8</td>
</tr>
<tr>
<td>a-zA-Z0-9!@#$%^&amp;*()[]_+-=+;:&quot;&quot;,./?</td>
<td>10</td>
</tr>
<tr>
<td>0-9a-f</td>
<td>32</td>
</tr>
</tbody>
</table>
TO THE CLOUD
WHY USE CLOUD

- Easy alternative to having dedicated cracking hardware
- Low to no setup cost
- Readily available images for deployment
- Scales as required
CRACKING WITH JTR

- Jumbo distribution
- Define dynamic format
- Distributed cracking with MPI
- Increase performance with CUDA or OpenCL
JTR DYNAMIC FORMAT

[List.Generic:dynamic_1011]
Expression=md5($s.$p.$s2) (Payment gateway signature)
Flag=MGF_SALTED
Flag=MGF_SALTED2
Func=DynamicFunc__clean_input
Func=DynamicFunc__append_salt
Func=DynamicFunc__append_keys
Func=DynamicFunc__append_2nd_salt
Func=DynamicFunc__crypt_md5
Test=$dynamic_1011$c4a5babae57a7d58610ce33ca79622c9$ABC9999|$$2|Invoice986616|100.00:xyz123

Validate:
./john --test --format=dynamic_1011
HTML FORM TO DYNAMIC HASH

my $html = eval { local $/; open my $fh, "$ARGV[0]"; return <$fh>; close($fh); }; $html =~ m/(<h3>Credit Card Payment.*?<\/form>)/ms; my $pwgform = 1; my $form = HTML::Form->parse($pwgform, 'file:///'); my $merchantID = $form->find_input('MERCHANDT_ID')->{value}; my $amount = $form->find_input('AMOUNT')->{value}; my $hash = $form->find_input('MD5HASH')->{value}; my $account = $form->find_input('ACCOUNT')->{value}; my $currency = $form->find_input('CURRENCY')->{value}; my $notifyurl = $form->find_input('SHOP_DOMAIN')->{value}; my $shopname = $form->find_input('SHOP_NAME')->{value}; my $orderID = $form->find_input('ORDER_ID')->{value}; my $floatAmt = $form->find_input('FLOAT_AMOUNT')->{value}; my $timestamp = $form->find_input('TIMESTAMP')->{value}; my $id_card = $form->find_input('ID_CARD')->{value}; my $lang = $form->find_input('LANG')->{value}; print \"$dynamic_1011\".$hash."$merchantID\"$\2$orderID$\amount$\currency\n";
DISTRIBUTED CRACKING WITH MPI

Update makefile:
CC = mpicc -DHAVE_MPI -DJOHN_MPI_BARRIER -DJOHN_MPI_ABORT
MPIOBJ = john-mpi.o

Setup MPI over ssh using key based authentication

Create a MPI host file
192.168.1.2 slots=2
192.168.1.3
CRACKING WITH GPU

- GPU greatly outperforms CPU for hash calculation
- Scales with devices
- CUDA or OpenCL
- Available through some cloud providers
## CRACKING WITH GPU

<table>
<thead>
<tr>
<th>CUDA</th>
<th>OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nvidia</td>
<td>Khronos group</td>
</tr>
<tr>
<td>Compiler builds kernel</td>
<td>Builds kernel at runtime</td>
</tr>
<tr>
<td>C language extensions</td>
<td>API only</td>
</tr>
<tr>
<td>Buffer offsets allowed</td>
<td>Buffer offsets not allowed</td>
</tr>
<tr>
<td>Pointer traversal allowed</td>
<td>Must use pointer arithmetic</td>
</tr>
</tbody>
</table>
LOTS OF OPEN SOURCE OPTIONS

- Jtr
  http://www.openwall.com/john/
- Cryptohaze Multiforcer
  http://www.cryptohaze.com/multiforcer.php
- Wisecracker
  http://selectiveintellect.com/wisecracker.html
- Whitepixel
  http://whitepixel.zorinaq.com/
- Defuse gpu cracker
  https://defuse.ca/gpucrack.htm
- OCLcrack
  https://github.com/sghctoma/oclcrack
<table>
<thead>
<tr>
<th>refundCard</th>
</tr>
</thead>
<tbody>
<tr>
<td>merchantUUID</td>
</tr>
<tr>
<td>apiKey</td>
</tr>
<tr>
<td>transactionAmount</td>
</tr>
<tr>
<td>transactionCurrency</td>
</tr>
<tr>
<td>transactionID</td>
</tr>
<tr>
<td>refundAmount</td>
</tr>
<tr>
<td>hash</td>
</tr>
<tr>
<td>queryCard</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>merchantUUID</td>
</tr>
<tr>
<td>apiKey</td>
</tr>
<tr>
<td>transactionID</td>
</tr>
<tr>
<td>hash</td>
</tr>
</tbody>
</table>
CONCLUSION

• Don’t rely on the browser to drive traffic between the merchant website and the payment gateway
• Crypto is hard
• Use more than one unknown variable in request validation
• Use a long secret
• Use token based redirection
• Protect all parameters used in the request
• Use an established keyed-hash message authentication code (HMAC)
• Weak request validation does not equal an exploitable vulnerability
THERE WILL ALWAYS BE IMPLEMENTATION BUGS
<?php
/** Constants */
$customer_data_dir = "/var/tmp";

$customer_ref = $_POST["customer_ref"];

if($customer_ref == null) {
    header("HTTP/1.0 404 Not Found");
} else {
    unlink("$customer_data_dir/$customer_ref.txt")
}
?>
// Execute the HTTPS post via CURL
$ch = curl_init($this->gateway_url);
curl_setopt($ch, CURLOPT_HEADER, 0);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, rtrim($this->field_string, ' '));

// Do not worry about checking for SSL certs
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 2);

$this->response_string = urldecode(curl_exec($ch));
PHP’S TYPE JUGGLING

```php
// Check to see if hashes match or not
if ($md5hash != $_POST['md5']) {
    $return = "BAD HASH";
}
elseif ($result == "00") {
```
Slides and demo code can be found at:

https://github.com/wireghoul/presentations/BHAsia2014
Contact details

BAE Systems Applied Intelligence
Suite 1, 50 Geils Court
Deakin ACT 2600
Australia
Tel: +61 1300 027 001
Fax: +61 2 6260 8828
Email: australia@baesystemsdetica.com
Web: www.baesystemsdetica.com.au