Dual Canonicalization

An Answer to the Homograph Attack

James Helfrich, PhD

Brigham Young University – Idaho

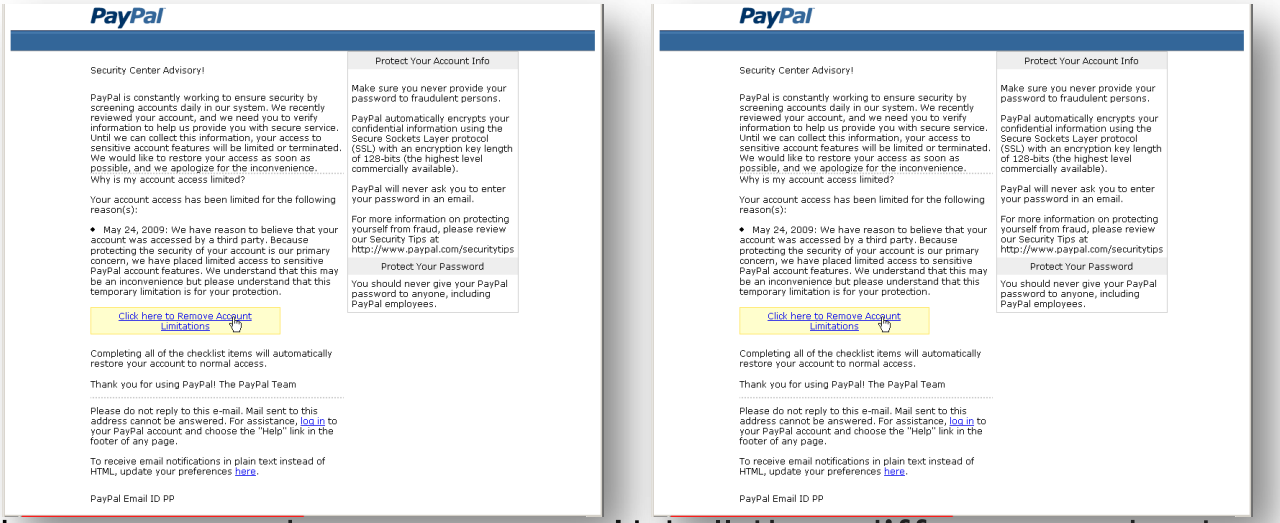
HelfrichJ@byui.edu

APWG

Unifying the
Global Response
to Cybercrime

Phishing

Phishing is defined as "the impersonation of reputable companies in order to induce individuals to reveal personal information, such as passwords and credit card numbers."



What happens when you can't tell the difference between a counterfeit site and the legitimate site?

http://www.microsoft.com

http://www.microsoft.com

Oxford University Press, 2009 [Online]. Available: http://dictionary.oed.com
Srikwan & Jakobsson, 2007 in *Oops… I clicked.* [Online]. Available: www.securitycartoon.com

Homographs

Homophone	Two words pronounced the same but have different meanings or spellings	hair hare
Homonym	Two words spelled the same but have different meanings	sow (noun) sow (verb)
Homograph	Two words or tokens that appear the same but are encoded differently	HTM (Latin) H T M (Greek) H T M (Cyrillic)

There are two variants of a homograph attack:

- **Counterfeit**: An attacker presents to an observer text that appears to be authentic but is actually counterfeit.
  Example: www.micr o s o ft.com
- **Bypass**: An attacker attempts to bypass a filter by presenting text a human can understand but a filter will not recognize.
  Example: "h.o.m.o.g.r.a.p.h"

E. Gabrilovich and A. Gontmakher, "The homograph attack," Communications of the ACM, vol. 45, no. 2, p. 128, February 2002.

Homograph Mitigation Techniques

Many strategies have been proposed to mitigate homograph attacks:

Prohibit mixing alphabets

If a URL consists of multiple alphabets, flag it as a homograph. This excludes many legitimate URLS and does not catch all the cases.

(Gabrilovich & Gontmakher; 2002, Krammer, 2006)

Heuristics

A scoring to compute the probability of an attack. Helpful in URL spoofing, but not generic. IRI/IDN SecuChecker, REGCAP, Quero

(Krammer, 2006; Yee et al, 2006; Liu et al, 2007; Fu, 2006))

Punycode

Identify text residing in a script not native to the user's browser (www.xn--hmgraph-8ofb.xn--cm-jbc". This exposes users of multi-script locales.

(Gupta, 2005)

Earth Mover's Distance

Compare the degree of visual similarity between glyphs using Earth Mover's Distance. Useful for counterfeit detection.

(Fu, Zhang, Deng, Wenyin, 2006; Fu, Deng, Wenyin, 2006)

Script Coloring

Highlight characters in each script with a distinct color. Problems: false positives (CNNenEspañol.com), not prominent enough.

(Wu et al, 2006; Krammer, 2006; Wenyin et al, 2008)

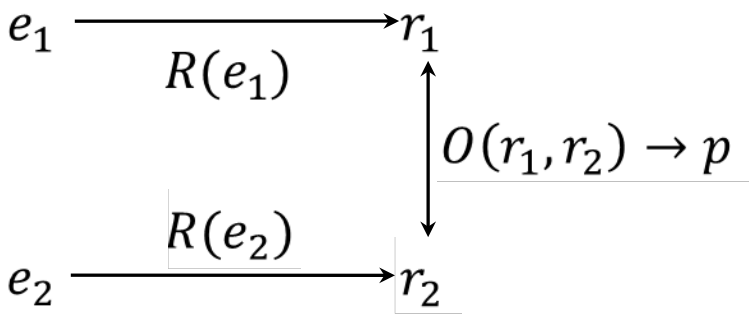
Kernel Density Estimation

Compute the degree of visual similarity between glyphs using Kernel Density Estimation. Also useful for counterfeit detection.

(Fu, Deng, Wenyin, & Little, 2006))

Definitions & Theoretical Framework (1 of 3)

e	*Encoding*: A representation of some presentation. Examples include Unicode for text, HTML for documents, MP3 for sound, and JPG for images.
r	*Rendition*: How a given encoding e appears to the observer. This could be pixels on the screen or music in the ears
$R(e_1) \rightarrow r_1$	*Rendering function*: How a given encoding is rendered or displayed into the rendition format. This could be an edit control in a browser's address textbox, Adobe Acrobat® reader, or a music player
$O(r_1, r_2) \rightarrow p$	*Observer function*: The probability that a given observer will consider two renditions the same. This probability is called the threshold of belief. An alternate form of this is $O(r_1, r_2, p)$ which returns true if the observer considers r_1 and r_2 the same at p probability and false otherwise.



Definitions & Theoretical Framework (2 of 3)

$H(e_1, e_2) \rightarrow p$	*Homograph function*: The probability that a pair of encodings will be perceived as being the same to an observer. An alternate form of this is $H(e_1, e_2, p)$ which returns true if the observer considers e_1 and e_2 the same at p probability and false otherwise. The homograph function is defined in terms of the observer function and the rendering function: $$H(e_1, e_2, p) = O(R(e_1), R(e_2), p)$$
h	*Homograph set*: A set of unique encodings perceived by the observer as being the same. In other words, two encodings e_1 and e_2 are considered homographs if an observer $O()$ considers them to be in the same set h. This definition can be represented mathematically with: $$\forall e_i, e_j \quad e_i \in h \wedge e_j \in h \leftrightarrow H(e_1, e_2) \geq p$$

Homographs are two or more encodings belonging to the same homograph set.

Anti-homographs are two or more encodings belonging to different homograph sets.

Definitions & Theoretical Framework (3 of 3)

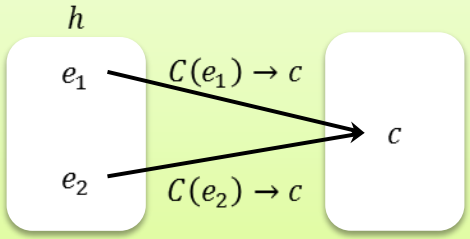
c	*Canon*: A canon is defined as a unique representation of a homograph set. Note that the format of the canonical token c may or may not be the same format as the encoding e or the rendition format r.
$C(e) \rightarrow c$	*Canonicalization function*: Canonicalization is "the process by which various equivalent forms of a name are resolved to a single, standard name – the canonical name" (Howard, 2002). We will define this canonicalization process with the function $C()$, taking an encoding e as input and returning a canonical token c.

The canonicalization function has two properties:

Reliable Canons Property

The canonicalization function will always yield identical canonical tokens for any homograph pair.

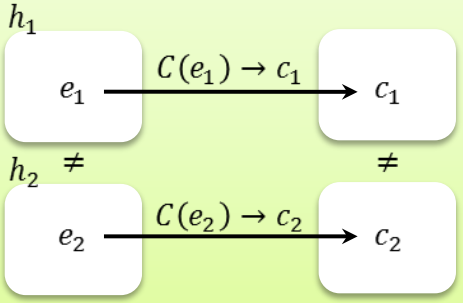
$$\forall\, e_1, e_2 \quad C(e_1) = C(e_2) \;\leftrightarrow\; H(e_1, e_2) \geq p$$



Unique Canons Property

Any pair of anti-homograph encodings will yield different canonical tokens.

$$\forall\, e_1, e_2 \quad C(e_1) \neq C(e_2) \;\leftrightarrow\; H(e_1, e_2) < p$$



M. Howard and D. LeBlanc, Writing secure code, Redmond, WA: Microsoft Press, 2002.
T. Hoad and J. Zobel, "Methods for identifying versioned and plagiarised documents," *J. Am. Soc. In. Sci. Techology,* vol. 54, no. 3

Dual Canonicalization

Dual Canonicalization is a technique for determining if two encodings are homographs by comparing the output of canonicalization functions. Three things must be known:

Rendering Function

One must understand how the encoding is rendered for the user. All aspects of how the encoding is rendered must be taken into account.

Observer Function

One must understand the degree of scrutiny the observer will consider the renditions. Must the renditions be *perfect*, *close*, or just *readable*?

Canonicalization Function

One must define a canonicalization function taking the rendering function and the observer function into account. This function must satisfy the Reliable Canons Property and the Unique Canons Property.

Experimental Results (1 of 3)

Experiment 1: Classic Unicode Attack

The attacker creates a counterfeit URL that appears like a legitimate URL yet consists of non-Latin character encodings. In this case, the rendering function is the Arial Unicode font rendered in a Unicode-enabled edit control in a browser window. We will initially set the observer scrutiny level to 100%, two presentations will be considered the same iff they are pixel-for-pixel identical. Thus the observer function $O(r_1, r_2,) \rightarrow p$ can be reduced to $r_1 = r_2$ because $p \rightarrow 1.0$.

Homographs

Encoding	Rendition
һomogｒарһ	homograph
hоmoɡｒaрһ	homograph
һoｍοgrａｐｈ	homograph
һｏmｏgraｐһ	homograph

Anti-Homographs

Encoding	Rendition
һоmｏΥrарһ	homoYraph
һоhοɡｒaрh	hohograph
ｈоmo８ｒаｐh	homo8raph
hоmｏｇｒарl	homɑgrapl

Apple Inc., "The TrueType Font File," 18 December 2002. [Online]. Available: https://developer.apple.com/fonts/TTRefMan/RM06/Chap6.html. [Accessed 24 May 2012].

Experimental Results (2 of 3)

Experiment 2: Near Homograph Attack

Next we relax the level of observer scrutiny to include the set of "near homographs" or the set of characters that the observer will *probably* recognize as the same. Here the rendering function remains the same but the observer function is relaxed. The canonicalization function will then be determined by how similar rendered glyphs appear to the user (visual edit distance or VED computed using the Knuth-Morris-Pratt algorithm) and the semantic differences between the characters (SED).

Homographs

Encoding	Rendition
hoⅿｏɡｒарｈ	hom o g r ap h
hoṃоｇｒаｐh	hoṃo g r a p h
hｏṃｏgｒａpһ	h o ṃ o g r a ph
һoｍoｇrаph	homo g raph

Anti-Homographs

Encoding	Rendition
ｈоmοｇˋаｐh	h omo g `a p h
ｈｏｍ＠grａｐһ	h o m@gr a p h
ｈοⅿοurарh	h omouraph
&#xff48o;ṃоｇｒａＳh	h oṃo g r a Sh

A. Fu, X. Deng, L. Wenyin and G. Little, "The methodology and an application to fight against unicode attacks," in *Symposium on Usable Privacy and Security (SOUPS)*, Pittsburgh, 2006.

Experimental Results (3 of 3)

Experiment 3: HTML Filter Avoidance Attack

SPAM filter-avoidance scenario is when the attacker attempts to pass a word ("Homograph") through a filter yet remain readable to the observer. In this case, the encoding format is HTML. The Canonicalization Function is:

1. HTML input is rendered in a browser
2. Image is converted to Unicode using OCR (www.free-online-ocr.com)
3. All white spaces and punctuation are removed
4. Unicode homographs are canonicalized using $UC\_SimList_{0.8}$

Homograph

Encoding	Rendition
`<img src=".\Images\h.gif"/>o&#x217f; <img src=".\Images\O.gif"/> <img src=".\Images\g.gif"/>r&#x430;&#x440;<img src=".\Images \h.gif"/>`	homoɡraph

Anti-Homograph

Encoding	Rendition
`<img src=".\Images\bigh.gif"/>om<img src=".\Images\o.gif"/><img src=".\Images\bigg.gif"/><img src=".\Images\bigr.gif"/><img src=".\Images\bige.gif"/><img src=".\Images\bigp.gif"/><img src=".\Images \bigh.gif"/>`	HomoGREPH

R. Cockerham, "There are 600,426,974,379,824,381,952 wasy to spell Viagra," 2007. [Online]. Available: http://cockeyed.com/lessons/viagra/viagra.html. [Accessed 26 April 2012].

Opportunities for Future Research

The Dual Canonicalization technique can be applied to a wide variety of non-traditional scenarios.

- **E-mail client plugin**: Imagine a canonicalization function recognizing certain stock e-mail messages (ex: pay-pal, bank statements, etc.). If the canons match and the links do not refer to the specified IP, it can be definitively flagged as phishing.
- **URL spoofing**: If all registered domain names are canonized, a definitive list of spoofed names can be identified. Furthermore, registrars can canonize requests and compared it against existing names to prevent spoofed URLs.
- **Plagiarism detection**: Encoding format is ASCII encoded or printed text. Rendition function is reading (comprehension). Canonicalization function is a fingerprinting process (Hoad & Zobel, 2003).

T. Hoad and J. Zobel, "Methods for identifying versioned and plagiarised documents," *J. Am. Soc. In. Sci. Techology,* vol. 54, no. 3, pp. 203-215, February 2003.

Questions

Source
Experiment 1
UC-SimList
Experiment 3
strcmp()

```
/************
 * HOMOGRAPH
 * A set of c
 ************
class Homogra
{
public:
   // constru
   HomographS
   HomographS
   // operato
   HomographS

   // add ima
   void addIm

   // inserti
   friend ost
   friend ist
      throw (

   // other c
   Character
   int
   Character
   Character

private:
   Character
   vector <Ch
};
```

```
/************
 * EQUIVALENC
 * A set of H
 ************
class Equival
{
public:
   Equivalenc

   Character
   HomographS

   Character
   string
   String

   bool equal
   bool equal
   {
      return
   }
   bool equal
   {
      return
   }
   bool Equal
   {
      return
   }

private:
   map <Chara
   map <Chara
};
```

```
/************
 * EQUIVALENC
 * Generate a
 ************
Character Equ
{
   Character
   if (key !=
      return
   else
      return
}

/************
 * EQUIVALENC
 * Generate a
 ************
string Equiva
{
   String sPl
   String sHo

   for (int i
      sHomogr

   return sHo
}
```

```
/************************************************
 * EQUIVALENCE :: EQUALS
 * Are two strings equal according to their homographs?
 ************************************************/
bool Equivalence :: equals(const String & us1,
                           const String & us2,
                           bool verbose)
{
   // compare the sizes
   if (us1.size() != us2.size())
   {
      if (verbose)
         cout << "different sizes: "
              << us1.size() << " : " << us2.size() << endl;
      return false;
   }

   // compare the tokens
   for (int i = 0; i < us1.size(); i++)
      if (getKey(us1.at(i)) != getKey(us2.at(i)))
      {
         if (verbose)
            cout << "different keys at position " << i + 1<< ". " << hex
                 << getKey(us1.at(i)) << " " << (char)getKey(us1.at(i))
                 << " : "
                 << getKey(us2.at(i)) << " " << (char)getKey(us2.at(i))
                 << endl;
         return false;
      }

   return true;
}
```

UC-SimList *p*=100%

```
0021    1:FF01:!        1:01C3:!        1:
0022    1:0022:"        1:FF02:"
0023    1:0023:#        1:FF03:#
0024    1:0024:$        1:FF04:$
0025    1:0025:%        1:FF05:%
0026    1:FF06:&        1:0026:&
0027    1:FF07:'        1:0027:'
0028    1:0028:(        1:FF08:(
0029    1:FF09:)        1:0029:)
002A    1:002A:*        1:FF0A:*
002B    1:002B:+        1:FF0B:+
002C    1:002C:,        1:FF0C:,
002D    1:02C9:\257     1:2574:?        1:
?        1:FF0D:-        1:00AD:\255     1
002E    1:2024:\267     1:00B7:\267     1:
.        1:02D9:?        1:FF65:?        1
002F    1:002F:/        1:FF0F:/
0030    1:0030:0        1:FF10:0
0031    1:0031:1        1:FF11:1
0032    1:0032:2        1:FF12:2
0033    1:0033:3        1:FF13:3
0034    1:0034:4        1:FF14:4
0035    1:0035:5        1:FF15:5
0036    1:FF16:6        1:0036:6
0037    1:0037:7        1:FF17:7
0038    1:FF18:8        1:0038:8
0039    1:0039:9        1:FF19:9
003A    1:2236::        1:003A::        1:
003B    1:FF1B:;        1:003B:;        1:
003C    1:003C:<        1:FF1C:<
003D    1:003D:=        1:FF1D:=
003E    1:FF1E:>        1:003E:>
003F    1:FF1F:?        1:003F:?
0040    1:0040:@        1:FF20:@
0041    1:FF21:A        1:0410:?        1:
```

UC-SimList *p*=80%

```
0021    1:FF01:!        1:01C3:!        1:0021:!        0.8706896:FF4C:l        \
 0.8706896:006C:l        0.8706896:0406:?        0.8706896:0049:I        0.8706\
896:0399:?        0.8706896:01C0:|        0.8706896:04C0:?        0.8706896:216\
0:?        0.8706896:FF29:I        0.8706896:217C:?        0.8611111:05C0:?    \
    0.8514851:00A1:\241     0.8415841:FF49:i        0.8415841:0069:i        0.8\
415841:0456:?        0.8415841:2170:?        0.8380952:1F77:?        0.8217822:\
0131:i        0.8217822:03B9:?        0.8207547:1F31:?        0.8207547:1F30:? \
       0.8:03AF:?
0022    1:0022:"        1:FF02:"
0023    1:0023:#        1:FF03:#
0024    1:0024:$        1:FF04:$
0025    1:0025:%        1:FF05:%        0.815562:2105:?
0026    1:FF06:&        1:0026:&
0027    1:FF07:'        1:0027:'
0028    1:0028:(        1:FF08:(
0029    1:FF09:)        1:0029:)
002A    1:002A:*        1:FF0A:*        0.9673913:2217:*
002B    1:002B:+        1:FF0B:+
002C    1:002C:,        1:FF0C:,        0.8709677:2019:\222     0.8709677:201A:\
\202
002D    1:02C9:\257     1:2574:?        1:002D:-        1:02CD:_        1:2576:\
?        1:FF0D:-        1:00AD:\255     1:2011:-        1:2010:-
002E    1:2024:\267     1:00B7:\267     1:002E:.        1:2027:?        1:FF0E:\
.        1:02D9:?        1:FF65:?        1:0387:?        0.8235294:02D1:?        \

002F    1:002F:/        1:FF0F:/
0030    1:0030:0        1:FF10:0        0.8293515:FF19:9        0.8293515:0039:\
9        0.808642:03B8:? 0.8033333:10DB:?
0031    1:0031:1        1:FF11:1
0032    1:0032:2        1:FF12:2        0.8919861:01BB:?
0033    1:0033:3        1:FF13:3        0.8060837:10D5:?
0034    1:0034:4        1:FF14:4
0035    1:0035:5        1:FF15:5        0.8086643:01BC:?
0036    1:FF16:6        1:0036:6        0.8862876:10DC:?        0.88:10DB:?    \
 0.8306189:0038:8        0.8306189:FF18:8        0.812709:10EA:? 0.8076923:10E8\
```

APWG

Unifying the
Global Response
to Cybercrime

Homographs *p*=100%

```
<html><span style='font-family:"Arial Unic
<p>&#x4bb;omog&#xff52;&#x430;&#x440;&#x4bb
<p>h&#x43e;mo&#x261;&#xff52;a&#x440;&#x4bb
<p>&#x4bb;o&#xff4d;&#x3bf;gr&#xff41;&#xff5
<p>&#x4bb;&#xff4f;m&#xff4f;gra&#xff50;&#x4
<p>&#xff48;&#x3bf;m&#x3bf;&#x261;&#xff52;a
<p>h&#x43e;m&#xff4f;&#xff47;&#xff52;&#xff4
<p>&#xff48;o&#xff4d;o&#xff47;ra&#x440;h</p
<p>&#xff48;&#xff4f;&#x217f;&#x3bf;&#xff47;
<p>hom&#x3bf;&#x261;&#xff52;&#x430;&#xff50
<p>&#x4bb;&#x43e;&#xff4d;&#xff4f;&#x261;r&
<p>&#x4bb;&#x43e;&#x217f;o&#x261;&#xff52;&
<p>ho&#x217f;o&#x261;&#xff52;&#x430;&#x440
<p>&#xff48;&#xff4f;&#x217f;&#x3bf;&#x261;r
<p>ho&#xff4d;&#x3bf;&#x261;r&#xff41;&#xff5
<p>h&#x43e;&#xff4d;og&#xff52;a&#x440;&#xff
<p>&#xff48;&#x43e;&#xff4d;&#x3bf;&#xff47;&
<p>&#x4bb;&#x43e;m&#x3bf;&#x261;&#xff52;a&
<p>&#xff48;&#x3bf;&#xff4d;&#x43e;&#xff47;r
<p>&#x4bb;&#xff4f;&#x217f;&#xff4f;&#xff47;
<p>h&#xff4f;m&#x3bf;g&#xff52;&#xff41;ph</p
<p>&#x4bb;&#x43e;m&#x43e;&#x261;&#xff52;&#
<p>&#xff48;om&#xff4f;&#x261;r&#x430;p&#x4b
<p>ho&#x217f;ogr&#xff41;&#xff50;&#x4bb;</p
<p>&#x4bb;&#x43e;mo&#x261;r&#x430;&#x440;h
<p>h&#x43e;m&#x3bf;&#xff47;&#xff52;a&#x440
<p>&#xff48;omog&#xff52;&#xff41;&#x440;&#xf
<p>h&#x3bf;mo&#xff47;r&#xff41;ph</p>
<p>&#x4bb;o&#xff4d;&#x43e;&#x261;&#xff52;a
<p>&#xff48;o&#xff4d;&#x43e;&#xff47;r&#x430
<p>h&#xff4f;&#xff4d;og&#xff52;&#x430;&#x44
<p>h&#x43e;&#x217f;o&#x261;&#xff52;&#x430;
<p>&#xff48;o&#x217f;&#x3bf;gr&#xff41;&#xff
<p>&#xff48;&#x3bf;m&#x3bf;&#xff47;r&#x430;
<p>&#x4bb;omo&#x261;r&#xff41;ph</p>
</span></html>
```

Anti-Homographs *p*=100%

```
<html><span style='font-family:"Arial Unicode MS","sans-serif"'>
<p>&#x4bb;&#x43e;m&#xff4f;&#x3a5;r&#x430;&#x440;&#x4bb;</p>
<p>&#x4bb;&#x43e;h&#x3bf;&#x261;&#xff52;a&#x440;h</p>
<p>&#xff48;&#x43e;mo&#xff18;&#xff52;&#x430;&#xff50;h</p>
<p>h&#x43e;&#x217f;&#xff4f;&#xff47;&#xff52;&#x430;&#x440;&#x217c;</p>
<p>h&#x43e;m&#x43e;&#xff47;&#xff52;&#xff38;ph</p>
<p>&#x4bb;&#xff4f;m&#xff37;&#xff47;r&#x430;&#x440;&#xff48;</p>
<p>&#xff48;&#x3bf;&#xff4d;&#xff4f;g&#xff52;A&#x440;h</p>
<p>&#x4bb;o&#xff4d;&#xff4f;gr&#xff41;&#xff41;&#x4bb;</p>
<p>&#x4bb;&#x43e;&#x217f;&#x43e;&#xff47;ra&#x399;&#xff48;</p>
<p>&#x4bb;&#xff4f;&#xff4d;&#x43e;gr&#xff41;&#xff4e;h</p>
<p>&#x4bb;&#x397;&#xff4d;&#x3bf;g&#xff52;&#x430;p&#x4bb;</p>
<p>&#x4bb;&#x3bf;p&#xff4f;graph</p>
<p>h&#x3bf;&#xff4d;5g&#xff52;&#xff41;&#x440;h</p>
<p>&#x4bb;&#x43e;m&#xff4f;&#xff3f;r&#x430;p&#xff48;</p>
<p>&#xff4b;om&#x43e;&#xff47;&#xff52;a&#xff50;&#xff48;</p>
<p>h&#x3bf;&#xff4d;&#x217f;&#xff47;&#xff52;&#x430;&#xff50;&#x4bb;</p>
<p>&#xff48;&#xff4f;&#x430;&#xff4f;&#xff47;&#xff52;&#x430;p&#x4bb;</p>
<p>&#x4bb;&#xff4f;&#xff4d;&#xff4f;&#x261;&#xff52;&#xff41;&#x212a;&#x4bb;</p>
<p>&#xff48;omog&#xff52;a&#x440;&#xff0a;</p>
<p>&#x4bb;&#xff4f;&#xff03;o&#xff47;rap&#xff48;</p>
<p>&#xff06;&#x3bf;&#xff4d;og&#xff52;aph</p>
<p>&#x4bb;om&#xff4f;&#xff03;ra&#x440;&#xff48;</p>
<p>&#xff48;o&#xff4d;&#x3bf;&#xff47;&#xff32;&#x430;p&#xff48;</p>
<p>&#x4bb;&#x43e;m&#x43e;&#xff47;Qa&#x440;&#xff48;</p>
<p>&#xff37;&#x3bf;m&#xff4f;&#xff47;r&#x430;&#x440;&#xff48;</p>
<p>h&#xff4f;&&#xff4f;g&#xff52;&#x430;&#x440;h</p>
<p>h&#x3bf;m&#x43e;&#x261;&#xff23;a&#x440;&#x4bb;</p>
<p>&#x4bb;&#xff3b;&#xff4d;&#x3bf;&#x261;r&#x430;&#xff50;&#xff48;</p>
<p>&#x4bb;o&#xff4d;&#xff4f;&#xff47;&#xff11;&#xff41;p&#xff48;</p>
<p>ho&#x443;o&#xff47;r&#xff41;&#x440;&#xff48;</p>
<p>h&#x3bf;&#xff4d;&#x3bf;&#x261;&#xff52;a&#x39d;&#x4bb;</p>
<p>&#xff48;&#x43e;&#xff4d;&#xff4f;g&#xff42;ap&#xff48;</p>
<p>&#xff4b;&#x43e;m&#x3bf;&#x261;ra&#x440;h</p>
<p>&#x4bb;&#x43e;&#x217f;o&#x440;r&#x430;ph</p>
</span></html>
```

APWG

Unifying the
Global Response
to Cybercrime

Homographs *p*=80%

```
<html><span style='font-size:20;font-famil
<p>&#x4bb;&#x43e;&#x1e43;&#x3bf;&#x261;<im
.\Images\biga.gif"/><img src=".\Images\big
p>
<p><img src=".\Images\h.gif"/><img src=".\
\Images\bigg.gif"/><img src=".\Images\r.gi
f"/>&#x4bb;</p>
<p>&#x4bb;<img src=".\Images\bigo.gif"/>mo
=".\Images\p.gif"/><img src=".\Images\bigh
<p><img src=".\Images\h.gif"/>&#x43e;<img
 src=".\Images\bigr.gif"/><img src=".\Imag
f"/><img src=".\Images\h.gif"/></p>
<p><img src=".\Images\h.gif"/><img src=".\
if"/>o<img src=".\Images\g.gif"/>r<img src
\Images\bigh.gif"/></p>
<p><img src=".\Images\h.gif"/>o&#x1e43;<im
.\Images\g.gif"/><img src=".\Images\bigr.g
x440;h</p>
<p><img src=".\Images\bigh.gif"/><img src=
m.gif"/><img src=".\Images\o.gif"/>g<img s
es\biga.gif"/><img src=".\Images\p.gif"/><
<p>&#x4bb;o&#x1e43;&#x43e;<img src=".\Imag
f"/><img src=".\Images\biga.gif"/>&#x440;&
<p><img src=".\Images\bigh.gif"/>&#x3bf;&#
g src=".\Images\r.gif"/><img src=".\Images
img src=".\Images\h.gif"/></p>
<p>h&#x43e;<img src=".\Images\m.gif"/>&#x3
g src=".\Images\biga.gif"/>&#x440;<img src
<p><img src=".\Images\h.gif"/>&#x3bf;<img
;<img src=".\Images\bigp.gif"/><img src=".
<p>h&#x3bf;&#x1e43;o&#x261;r<img src=".\Im
igh.gif"/></p>
<p><img src=".\Images\h.gif"/>o&#x217f;&#x
g src=".\Images\r.gif"/><img src=".\Images
</p>
</span></html>
```

Anti-Homographs *p*=100%

```
<html><span style='font-size:20;font-family:"Arial Unicode MS","sans-serif"'>
<p>h&#x43e;<img src=".\Images\m.gif"/>&#x3bf;<img src=".\Images\bigg.gif"/>r<im\
g src=".\Images\biga.gif"/>ph</p>
<p>&#x4bb;<img src=".\Images\o.gif"/>&#x217f;&#x3bf;g<img src=".\Images\bigr.gi\
f"/>a&#x440;<img src=".\Images\h.gif"/></p>
<p><img src=".\Images\h.gif"/><img src=".\Images\bigo.gif"/><img src=".\Images\\
m.gif"/>o&#x261;<img src=".\Images\bigr.gif"/>a<img src=".\Images\p.gif"/>&#x4b\
b;</p>
<p>h&#x43e;<img src=".\Images\bigm.gif"/><img src=".\Images\bigo.gif"/><img src\
=".\Images\g.gif"/><img src=".\Images\r.gif"/>&#x430;<img src=".\Images\bigp.gi\
f"/><img src=".\Images\bigh.gif"/></p>
<p><img src=".\Images\h.gif"/><img src=".\Images\bigo.gif"/>m<img src=".\Images\
\o.gif"/><img src=".\Images\g.gif"/><img src=".\Images\r.gif"/><img src=".\Imag\
es\biga.gif"/><img src=".\Images\p.gif"/>&#x4bb;</p>
<p>h<img src=".\Images\bigo.gif"/>m&#x43e;<img src=".\Images\g.gif"/><img src="\
.\Images\bigr.gif"/><img src=".\Images\biga.gif"/><img src=".\Images\bigp.gif"/\
>&#x4bb;</p>
<p><img src=".\Images\h.gif"/>&#x43e;<img src=".\Images\m.gif"/>&#x43e;<img src\
=".\Images\g.gif"/><img src=".\Images\bigr.gif"/><img src=".\Images\biga.gif"/>\
ph</p>
<p>&#x4bb;<img src=".\Images\bigo.gif"/>m&#x43e;<img src=".\Images\bigg.gif"/><\
img src=".\Images\bigr.gif"/><img src=".\Images\a.gif"/><img src=".\Images\p.gi\
f"/><img src=".\Images\h.gif"/></p>
<p>h<img src=".\Images\o.gif"/><img src=".\Images\m.gif"/>&#x3bf;gra<img src=".\
\Images\bigp.gif"/>h</p>
<p><img src=".\Images\bigh.gif"/>&#x43e;&#x1e43;<img src=".\Images\o.gif"/>g<im\
g src=".\Images\r.gif"/><img src=".\Images\a.gif"/><img src=".\Images\bigp.gif"\
/>h</p>
<p><img src=".\Images\h.gif"/>o<img src=".\Images\bigm.gif"/><img src=".\Images\
\bigo.gif"/>&#x261;<img src=".\Images\r.gif"/><img src=".\Images\biga.gif"/><im\
g src=".\Images\bigp.gif"/><img src=".\Images\bigh.gif"/></p>
<p><img src=".\Images\bigh.gif"/>om&#x43e;<img src=".\Images\bigg.gif"/><img sr\
c=".\Images\r.gif"/>a<img src=".\Images\p.gif"/>&#x4bb;</p>
<p><img src=".\Images\bigh.gif"/><img src=".\Images\bigo.gif"/><img src=".\Imag\
es\bigm.gif"/>&#x3bf;gr<img src=".\Images\biga.gif"/>&#x440;<img src=".\Images\\
h.gif"/></p>
```

APWG

Unifying the
Global Response
to Cybercrime

```
bool operator == (const string & s1,
                  const string & s2)
{
   const char *p1;
   const char *p2;

   // for each member of the two strings
   for (p1 = s1.c_str(), p2 = s2.c_str();
        *p1 && *p2;
        p1++, p2++)
   {
       // convert to lowercase
       char c1 = tolower(*p1);
       char c2 = tolower(*p2);

       // compare the lowercase versions
       if (c1 != c2)
           return false;
   }

   // success only if both are the same
   return true;
}
```

Consider the standard string compare problem. If there are n letters in the string, then there are n^2 possible combinations:

security
securitY
securiTy
securiTY
securIty
securItY
…

Rather than compare all these combinations, it is much more efficient to compare the lowercase version of each.