Enhanced CAPTCHAs: Using Animation to Tell Humans and Computers Apart

Elias Athanasopoulos and Spiros Antonatos

Institute of Computer Science, Foundation for Research and Technology Hellas,
P.O Box 1385 Heraklio, GR-711-10 Greece
{elathan, antonat}@ics.forth.gr

Abstract. Completely Automated Public Turing Test to tell Computers and Humans Apart (CAPTCHA) is a –rather– simple test that can be easily answered by a human but extremely difficult to be answered by computers. CAPTCHAs have been widely used for practical security reasons, like preventing automated registration in Web-based services. However, all deployed CAPTCHAs are based on the static identification of an object or text. All CAPTCHAs, from simple ones, like typing the distorted text, to advanced ones, like recognizing an object in an image, are vulnerable to the *Laundry* attack. An attacker may post the test to a malicious site and attract its visitors to solve the puzzle for her. This paper focuses on sealing CAPTCHAs against such attacks by adding a dimension not used so far: animation. Animated CAPTCHAs do not have a static answer, thus even when they are exposed to laundering, unsuspected visitors will provide answers that will be useless on the attacker's side.

Keywords: Web Security, CAPTCHA, Laundry attacks

1 Introduction

CAPTCHAs[7] are challenge-response puzzles used to determine whether a user is human or not. There are several types of CAPTCHA tests, including distorted text, pictures of objects or even audio clips in case of impaired users. A simple example of a CAPTCHA test is shown in Figure 1. Users are requested to type the text displayed in the picture, "smwm" in our example. Most advanced examples, like the one in Figure 2, ask the user to identify an object, a person or an animal.

CAPTCHA tests are dynamically generated by computers, in contrast to the standard Turing test which is administered by a human. This characteristic allows them to be widely used for practical security reasons. Their applications span across multiple domains, from preventing worms and spam to online polls and search engines. The most common application of CAPTCHA tests is the prevention of automatic registration in Web-based services, like Web-based e-mail. E-mail provider sites include a CAPTCHA test as a step of the registration process to stop bots from subscribing and using their resources for spam distribution. Other applications involve online polls, where CAPTCHAs ensure that

only humans vote or Web-blogs, where CAPTCHAs protect the blog from the massive insertion of garbage content by automated scripts. CAPTCHA tests can

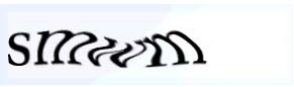

Fig. 1. An example CAPTCHA, which is solved if a user recognizes the word 'smwm'.

Fig. 2. A sophisticated CAPTCHA, which is solved if a user identifies all three animals.

Fig. 3. A modern CAPTCHA, which adds the distortion to the image.

be circumvented in several ways. Advanced character recognition programs[13] can extract the text from simple tests like the one in Figure 1. However, tests used today are not that simple. By adding noisy backgrounds, colours and increasing the level of distortion, tests become resistant to character recognition programs. An example of how modern CAPTCHA tests look like is shown in Figure 3. Apparently, all CAPTCHA tests are vulnerable to laundry attacks. An attacker may post the test to her site and lure the visitors of this site to solve the test for her, e.g. by providing free access to content after the test is solved. Laundry attacks are independent of the complexity of current CAPTCHA tests. Their key property is that they use the intelligence of a human, thus any CAPTCHA tests in their current form are vulnerable to this attack.

In this paper, we present a novel technique for preventing laundry attacks for CAPTCHAs. The key idea behind our approach is that the answer of the CAPTCHA is embedded inside the test, animated to avoid static properties of current tests.. All current forms of CAPTCHAs follow the "type the answer" pattern, which dramatically helps laundering. We propose another form of CAPTCHA, where the answer is not static but floats around the test.

Specifically, our approach is a test where various objects are randomly moving inside the test. One of them is the correct answer and the user has to click on it to complete the test. Animation does not prevent the user to tell the attacker *what* is the answer, but prevents her from telling the attacker *where* is the answer.

2 Background

CAPTCHAs were originally developed by AltaVista. They were used to block or discourage the submission of URLs to their search engine. In 2002, Baird et al. developed PessimalPrint, a CAPTCHA that uses a model of document image degradations that approximates ten aspects of the physics of machine-printing and imaging of text. Their model included spatial sampling rate and error, affine

spatial deformations, jitter, speckle, blurring, thresholding, and symbol size. BaffleText by PARC research uses non-English pronounceable character strings to defend against dictionary-driven attacks, and Gestalt-motivated image-masking degradations to defend against image restoration attacks.

Considerable research effort has been spent on breaking CAPTCHAs. Mori and Malik [13] have developed efficient methods based on shape context matching that can identify the word in an EZGimpy image with a success rate of 92%. Chellapilla et al.[10] have recently shown that computers are as good as or better than humans at single character recognition under all commonly used distortion and clutter scenarios used in todays CAPTCHAs. Poorly implemented CAPTCHAs can be also broken without using character recognition software but by exploiting session management weaknesses.

3 Animated CAPTCHAs

This Section aims at examining the threat model against current technologies used in the construction of CAPTCHAs and at describing our approach. To be fair, we give the attacker all benefits and we make no assumptions about the design of our approach. The key objectives of our approach is a) ease of deployment, we use industry standard technologies, such as Sun's Java Applets or Macromedia's Flash Movies, b) the test must be solvable by any user, at least as easy as current tests and finally c) robustness against attacks.

3.1 Laundry Attacks

Most services introduce a CAPTCHA to prevent automatic registration or ensure that a human is using them. Their main objective is to stop attackers from instrumenting their bots to automatically use the service for malicious purposes. An example is a Web e-mail service. In the absence of CAPTCHAs, the attacker could instruct her bots to register automatically to the service and start using the service in her way. For example, she can use the registered e-mail addresses to send spam. A CAPTCHA based on a static image is frequently used by large e-mail providers, such as Microsoft, Google and Yahoo, to ensure that the registration process was completed by a human and not a bot.

One way to defeat CAPTCHAs, based on a static image, such as the one in Figure 1, is by using sophisticated pattern matching. A bot can run special pattern recognition software that identifies the distorted word and eventually solve the test. However, the complexity of a static image CAPTCHA can be easily augmented and thus make the task of the pattern recognition program quite harder. Such an example can be seen in Figure 3. Although a human can easily identify that the "plus" word is displayed, the distortion in the picture increases exponentially the difficulty for a pattern recognition software. Thus, attackers are left with one solution to automatically solve a CAPTCHA: the laundry attack.

A laundry attack takes advantage of unsuspected users who will eventually solve a CAPTCHA in favor of the attacker, while they think that the CAPTCHA is solved for their own service. In more detail, consider an attacker who runs a popular Web site. Although it is out of the scope of this paper to explain how the malicious site will gain enough popularity, we can refer the reader to techniques[3].

Every visitor of the malicious site is lured to solve a CAPTCHA. However, this test is not generated by the site itself but it is actually the test of the victim service, for example the CAPTCHA of the registration phase of the Web e-mail service. No matter how difficult the test is, the answer now comes from a human and it is highly probable that it will be correct. The unsuspected user solves the test and the answer is "forwarded" to the victim service. In case the malicious site does not have enough popularity, it may present an aggressive behavior and periodically ask the user to solve a test, for example it can ask the user to solve one CAPTCHA per file download. In this way, the attacker has achieved to automatically solve CAPTCHAs independently of their difficulty, with a number of mistakes proportional to the number of mistakes ordinary users make and linearly to the number of visitors to her site.

The laundering of a CAPTCHA can be implemented by using the bots as intermediates. The malicious page that holds the "victim" puzzle contains a URL in the form "http://one_of_my_bots_IP/test.jpg". When an unsuspected user requests this URL, the bot sees the request and initiates the communication with the victim site, for example loads the registration page. Libraries like cURL [1] can be used to load pages from the command line offering full functionality similar to browsers, like cookies or redirections. The bot can also run a minimal HTML rendering engine and examine the loaded page, in order to spot the location of the CAPTCHA. Most sites have a constant name for the CAPTCHA image or even when they use dynamic names, their location inside the page is fixed or their names follow a specific pattern. After the test is located, it is then copied to "test.jpg" and served to the user. The user then answers the test to a form of the malicious site that has "http://one_of_my_bots_IP/submit.php" as action. The bot receives the answer and completes it to the loaded page. To the best of our knowledge, most services we tested allow multiple registrations per IP address, thus the attacker does not need to use all her bot power to perform the automated registrations, but a small fraction of it.

3.2 Animated CAPTCHAs to Prevent Laundry Attacks

Current forms of CAPTCHAs are subject to laundry attacks because of their static nature. They are pictures that contain the puzzle and the user has to complete the answer to a text field *outside* the puzzle. That is, the solution of the CAPTCHA is static and can be transfered between nodes of a malicious infrastructure (i.e. between a bot and a cooperative Web site which serves the laundry attack).

The first step we need to take is to transform a CAPTCHA test from a *static picture* to a *dynamic application*. That is, the answer must be completed *inside* the puzzle.

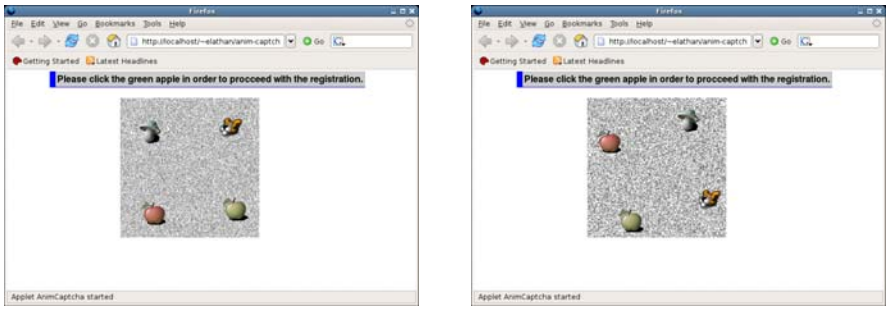

Fig. 4. An animated CAPTCHA, which we developed as a Java Applet.

Consider a test where the user has to identify an object. The test is now a mini-application that contains both the image and the form where the user submits the answer. The form points directly to the service that originally creates it, for example the Web e-mail service, and the puzzle is immutable, e.g. the attacker cannot change its forms to redirect them to her bots. We see how we can achieve immutable tests in Section 4. The attacker now has to launder the advanced test. When an unsuspected user completes her answer inside the puzzle, her submission will go directly to the victim site and will fail. There are mainly two reasons for the failure. First, the solution was submitted with a different cookie or session ID as the request (some services use the PHP session ID to map the answer to the request). Second, the solution was submitted from a different IP address than the request.

The attacker can circumvent this type of puzzles by posting a message like "Do not complete the answer inside the puzzle but to this text field". If the user follows the attacker's instructions (and we believe will do as she will have the incentives, e.g. access to the content) the same procedure as static puzzles can be followed. The second step is to eliminate the need for the user to type the answer and ask her to perform an action in the context of the application CAPTCHA. That is, the user is asked to *click* to the correct solution. The puzzle now contains multiple possible answers and the user has to click on the correct one. The click will submit the answer to the originating site. Again, the attacker may post a message like "Do not click on the test but complete this field where you would click". Although, the probability of a false answer has increased significantly (users may provide naive answers like "left" or "bottom") the attacker may assist users using Javascript snippets that can show the user the mouse coordinates.

The last step is to randomly animate the possible answers. While in the previous case answers remained static inside the puzzle, they now follow a random

path. Even when the unsuspected user tells the attacker where she clicked, this information is useless at the attacker's side. The animation of the puzzle, which runs at the bot, is completely different than the one which runs at the user's side. Thus, an answer like "I clicked on x,y coordinates" is useless as in that location it can be any answer when the bot clicks. In Section 4 we will discuss in more detail randomness issues. Animated CAPTCHAs succeed to force attackers to try conventional methods of breaking the test, like brute-force or reverse engineering attacks (see Section 5) and not use the human intelligence of unsuspected users.

4 Implementation

As CAPTCHAs are mainly used in Web sites, in this Section we will focus on how we can implement animated CAPTCHAs for browsers. Our goal is to construct a CAPTCHA that cannot be modified by the attacker. Two possible implementation approaches are Flash movies and Java applets. Both of these types are widely available and can be found at most browsers. We implemented our prototype using Java technology.

We want to prevent the attacker from two actions: (a) the attacker must not be able to identify the correct answer via reverse engineering, (b) as answers trigger a communication with the originating service, the communication endpoint must not be circumvented.

4.1 Reverse Engineering

Although we can hide the source code for an animated CAPTCHA, it can be decompiled for source inspection and modification using freeware tools[5]. Assuming that the answer of the CAPTCHA is embedded in the application, a decompilation process could reveal it to the attacker. Our first option, towards this direction, is the use of code obfuscation freeware tools[9].

However, it is well known that a system can not base its security strength in obfuscation or secrecy[12]. But, in our case, we want to avoid fast and automated reverse engineering, which will not require human interaction.

In more detail, a Web site will generate an animated CAPTCHA during every registration process (or other activity, which must be verified that it is used by a human). Generation of an animated CAPTCHA means that the CAPTCHA will be compiled from a standard template, will be randomized by inserting random code and, finally, it will be implanted with the correct solution and the session ID of the host requesting the service. The correct solution is considered also a unique per CAPTCHA random token. In the Web site's side, we assume that there is a storage component to keep the mapping between correct tokens and session IDs. After the generation of the CAPTCHA, the resulting Java class file will be obfuscated. If the whole process of the generation and obfuscation of CAPTCHAs is considered a heavy job for the server to perform it on demand,

it can use a pool of pre-generated CAPTCHAs (this pool can be maintained in low-traffic hours, in parallel with other maintenance procedures).

The above procedure guarantees that each animated CAPTCHA is a unique application. Each successful reverse engineering attempt, which should be also considered hard, must be triggered from a human and not by an automated program, since each CAPTCHA will have a different decompilation result. But, solving an animated CAPTCHA via human intervention is the definition of the CAPTCHA. The reverse engineering effort can not be re-used to solve automatically a collection of animated CAPTCHAs.

These ideas are already used in the case of polymorphic worms[14]. Code randomization has also been explored in various levels of software engineering and has been used to Computer Security[11].

4.2 Communication Circumvention

Our second goal is to protect the communication endpoint. Recall that the animated CAPTCHA is a Java applet, which embeds the token that maps in the solution and the session ID that maps to the host, which triggered the CAPTCHA. We need to prevent an attacker from locating the token or the session ID mechanically, since in that way she can create artificially a correct response to the service provider, and thus bypass the CAPTCHA.

In order to deal with this issue the token of the correct answer and the session ID must be encrypted and located in a non-fixed place of the Java bytecode. We can achieve the latter using code randomization as explained in 4.1. The decryption key should also placed in a random location of the Java bytecode. An attacker can still reveal the decryption key, as well as spot the encrypted token and session ID through reverse engineering, but as we have already explained this process must be repeated for every CAPTCHA instance, since each CAPTCHA is a unique Java applet (in terms of bytecode). Thus, it is still impossible for an automated program to circumvent the communication channel without the human interaction.

5 Attacks against Animated CAPTCHAs

With Laundry attacks eliminated, as it was described in Section 3.2, the malicious user will try to attack on the animated CAPTCHA itself. In this Section, we analyze in detail how an animated CAPTCHA can cope with attacks focused in animated CAPTCHAs. Furthermore, we implemented an actual animated CAPTCHA as a Java Applet (Figure 4), which had four objects following a circular orbit as possible solutions. During the implementation we tuned up various parameters in order to make the animated CAPTCHA more resistible against the attacks we describe below.

5.1 Brute Force Attack

The attacker may try to instruct her bot to continuously click on the puzzle until a possible answer is clicked. In that case, the probability to click a correct answer is $1/|possibleanswers|$. The number of possible answers cannot be high enough due to space reasons inside the puzzle. By placing tens of possible answers inside a limited space, the user will get confused and eventually she will be discouraged. Assume a bot-power (BP) of one thousand compromised machines and an animated CAPTCHA with ten possible answers. The probability for one member of the bot of solving randomly the CAPTCHA, $P(a)$, is 1/10. Assume, also, that the under attack site allows a maximum of five retries (R) per IP address per day and only one registration per IP. We can estimate the amount of puzzles the attacker may solve in one day: $P = BP*P(a)*R = 1000*1/10*5 = 500$. That is, the attacker may succeed to solve 500 puzzles per day, and thus complete automatically 500 registration processes, which is considered too high.

In order to cope with the brute force attack, we need to reduce the probability $P(a)$. We can easily transform the answer space of the puzzle from a discrete to a contiguous one. This can be done, by treating *every* click as an answer, including the clicks that reached the blank space, which surrounds the animation. Thus, the probability of clicking the correct answer now depends on two factors: an answer is found under the point where bot clicks and this answer is correct. Thus the probability to solve the puzzle by random clicking is now equal to the ratio r, where r is the surface area of one answer divided by the surface area of the whole puzzle area. The animated Java applet we developed depicted in Figure 4 occupies a surface of 480x480 pixels and each possible answer is an icon, which occupies a surface of 48x48 pixels. That is the ratio r is 0.01. Using the same parameters as before and $P(a) = 1/10*r = 0.001$, the attacker can automatically solve 5 puzzles a day. By tuning the ratio, we can achieve one to two orders of magnitude reduction on the number of puzzles that can be automatically solved and have a user-friendly puzzle at the same time.

Moreover, by combining more animated CAPTCHAs the probability $P(a)$ is reduced drammaticaly and not in a linear fashion. For example, the test may require the user to click a group of moving animals in a specific order based on their size. If the animated CAPTCHA has ten moving objects and three of them are animals, then the probability of clicking an animal is $P(a) = 0.001$. The probability of solving the test is the product of the three individual probabilities of clicking an animal. That is $P(a) = (0.001)^3$.

5.2 Remote Control Attack - Sweatshop Attack

The attacker may proceed with a manual installation on each bot, through display redirection techniques. Sweatshops are also an available option [6]. An attacker can hire employees from a sweatshop who will proceed to manual installation on the bots. Employees connect to each bot and redirect their display to a local machine. Specialized software can be used for display redirection. We experimented with VNC[8]. VNC is a lightweight display redirector that is ported

to most operating systems and can be easily installed in any system. We also considered the built-in functionality of the X11 server and the remote assistance feature of Windows XP professional. However, the X11 environment can be only found in Unix systems but most bots are Windows XP systems[15]. Furthermore, more bots are likely to have Windows XP Home installed. We ignore the fact that when VNC or remote assistance run, their presence is noticeable to the actual owner of the bot.

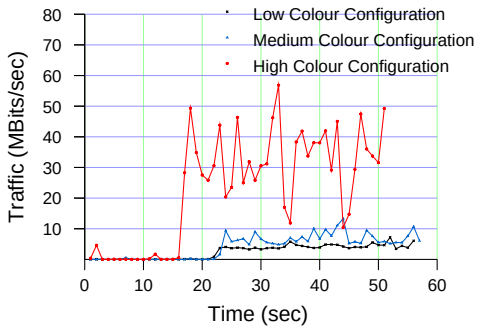

Fig. 5. VNC traffic for the various colour configurations. During the first 10 seconds we monitored the line without a VNC connection. The next 10 seconds we monitored the line with a VNC connection, but without running the CAPTCHA applet. During the rest of the period, the CAPTCHA applet runs remotely. Denote that we plot only the downstream traffic.

A way to defend against the sweatshop attack is to enhance the animation, and thus increase the bandwidth needed for the display redirector software. For example, observe that the animated CAPTCHA depicted in Figure 4 has a constantly changing background (similar to the snow effect of a non-working television) and that a multicolour display is required, since the user needs to be able to distinguish the green from the red apple. That is, a display redirector, configured in monochrome mode and in low resolution, so as to reduce the needed bandwidth cannot be used. Unfortunately, this choice has a drawback for people suffering of colour blindness, since they are not able to distinguish different colours[1]. However, W3C argues for the inaccessibility of all Visual CAPTCHAs for people with low vision[4].

In addition, if the answers inside the puzzle are animated slowly, the display redirection tools may be able to catch up with differences and display them correctly to the employees' machine.

We experimented on the delay introduced by VNC and bandwidth consumption for different display configuration between a hypothetical compromised ma-

[1] It is estimated that people suffering of colour blindness are almost 7% of all humans.

chine and a hypothetical machine owned by an employer of a sweatshop. Both machines were interconnected in a LAN with 100 MBit/sec network connection. We collected three traces, using the Ethereal tool[2], for a colour display of 64 (Low Colour Configuration), 256 (Medium Colour Configuration) and 24-bit (High Colour Configuration), respectively. In each experiment we enabled the compression of the transmitted data, supported by VNC. Although, VNC supports an even lowest display configuration, with 8 colours, we did not collect a trace, since our CAPTCHA was impossible to be solved: the green apple (the correct answer) was displayed with a yellow colour.

The results are plotted in Figure 5. Observe that even at the lowest colour configuration, VNC introduces a network traffic closed to 6 MBit/sec in order to display the animated CAPTCHA. Denote, that the VNC connection is fired up after the first 10 seconds have elapsed, but the CAPTCHA is launched after the first 20 seconds have elapsed. That is, the enhanced animation of the CAPTCHA causes the VNC server to transmit more information in the VNC client.

Based on Figure 5, we understand that an attack based on the remote control of a compromised machine may succeed only if the compromised machine is equipped with a network connection closed to 6 MBit/sec or better. On the other hand, it is almost trivial to modify various properties in order to make the remote control of an animated CAPTCHA harder to be achieved.

Someone can argue that existing commodity network speeds may increase in the near future and thus remote control software will have the required bandwidth to transmit a complex animation. However, no matter the speed, someone can create a high quality multimedia CAPTCHA, including motion and sound information dependent with the solution, which will need enormous bandwidth in order to be solved using a remote desktop application. Apparently, a multimedia CAPTCHA that combines high quality motion, high quality sound and a possible sequence of logic actions, is out of the context of this paper, but is subject for our future work.

	Property	Attack	Action	Result
1	Colourful answer	Remote Control	$\uparrow$	Bandwidth consumption $\uparrow$
2	Animated background	Remote Control	Enhance	Bandwidth consumption $\uparrow$
3	Dimension of background	Brute Force	$\uparrow$	$P(a) \downarrow$
4	Dimension of answers	Brute Force	$\uparrow$	$P(a) \downarrow$
5	Background is an answer	Brute Force	$+$	$P(a) \downarrow$
6	Frame delay	Remote Control	$\downarrow$	Bandwidth consumption $\uparrow$
7	Random orbit	Remote Control	$+$	Remote user difficult to adapt
8	Random frame delay	Remote Control	$+$	Remote user difficult to adapt
9	Code obfuscation	Laundry/RE	$+$	Reverse engineering effort$\uparrow$
10	Code randomization	Laundry/RE	$+$	Reverse engineering effort$\uparrow$
11	On the fly compilation	Laundry/RE	$+$	Reverse engineering effort$\uparrow$

Table 1. A list of properties of an animated CAPTCHA, which can be easily tuned in order to make the CAPTCHA more resistible in possible attacks.

In Table 1 we summarize various parameters that someone can modify and make an animated CAPTCHA resistible in the attacks we presented, namely the basic Laundry, the Reverse Engineering, the Brute Force and the Remote Control/Sweatshop attack. Denote that the symbol ↑ means 'increasing', the symbol ↓ means 'reduced' and the symbol + means 'adding'. For example, consider Property 4, which is translated as: "Increasing (↑) the dimension of answers, during a Brute Force attack, the probability of a random guess $P(a)$ is reduced (↓)".

6 Conclusion and Future Work

In this paper, we investigated the state of the art of possible attacks against CAPTCHAs; puzzles that try to distinguish a human from a computer program, used mainly to prevent a service to malicious programs, such as bots. We introduced a new form of a CAPTCHA, which is based on animation. We argued that animated CAPTCHAs can resist to modern attacks, like laundering, and common attacks, such as brute-force or reverse engineering. In regards to well organized attacks via sweatshops, we measured the traffic required by a popular remote desktop software to run our animated CAPTCHA prototype remotely and showed that commodity user equipment is not sufficient. Finally, we suggested various enhancements, which will burden the task for an attacker to bypass an animated CAPTCHA, using either of the possible attacks and forcing her to manually solve the puzzles.

We believe that our enhanced with animation CAPTCHA technology can resist in sophisticated attacks better than standard CAPTCHAs based on static images with distorted text. However, we have not exposed our technology to the users and get their feedback, in order to understand the possible complexity which is introduced to ordinary Web surfers. Thus, we plan to evaluate animated CAPTCHAs against static CAPTCHAs and see how user-friendly our technology is, by performing experiments where real users must register to a service using a process that embeds static and animated CAPTCHAs.

In addition, part of our future work is multimedia CAPTCHAs. Puzzles that embed high quality motion, sound and a solution that is the result of a sequence of logic actions.

Ackownledgments. We would like to thank the anonymous reviewers for their valuable comments. We also thank Kostas G. Anagnostakis (I2R) for his insightful comments. This work was supported in part by the project CyberScope, funded by the Greek General Secretariat for Research and Technology under the contract number PENED 03ED440, and by the FP6 project NoAH, funded by the European Union under the contract number 011923. Elias Athanasopoulos and Spiros Antonatos are also with the University of Crete.

References

1. cURL. http://curl.haxx.se/.

2. Ethereal. http://www.ethereal.com.
3. Google bombing. http://en.wikipedia.org/wiki/Google_bomb.
4. Inaccessibility of CAPTCHA, Alternatives to Visual Turing Tests on the Web. http://www.w3.org/TR/turingtest/.
5. JCavaJ Java Decompiler. http://www.bysoft.se/sureshot/jcavaj/index.html.
6. Sweatshop. http://en.wikipedia.org/wiki/Sweatshop.
7. The CAPTCHA Project. http://www.captcha.net/.
8. VNC. http://www.realvnc.com.
9. yGuard. http://www.yworks.com/en/products_yguard_about.htm.
10. Kumar Chellapilla, Kevin Larson, Patrice Simard, and Mary Czerwinski. Computers beat humans at single character recognition in reading based human interaction proofs (hips). In *Second Conference on Email and Anti-Spam (CEAS)*, 2005.
11. Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering code-injection attacks with instruction-set randomization. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 272–280, New York, NY, USA, 2003. ACM Press.
12. A. Kerckhoffs. La cryptographie militaire. In *Journal des Sciences Militaires, 9 Jan 1883, pp. 5-38*. http://www.petitcolas.net/fabien/kerckhoffs/.
13. G. Mori and J. Malik. Recognizing objects in adversarial clutter – breaking a visual captcha. In *Conf. Computer Vision and Pattern Recognition, Madison, USA*, June 2003.
14. Peter Szoer and Peter Ferrie. Hunting for metamorphic. In *Virus Bulletin Conference*, September 2001.
15. The Honeynet Project Whitepapers. Know your enemy: Tracking botnets, March 2005. http://www.honeynet.org/papers/bots/.