SRIKANTH KANDULA

surviving DDoS attacks

Srikanth Kandula is a graduate student at the MIT Computer Science and Artificial Intelligence Laboratory in the Networks and Mobile Systems group. His research interests are in networked systems and security.

kandula@MIT.EDU

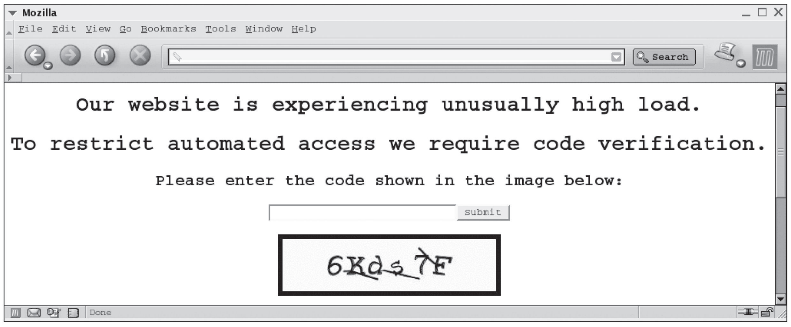
This article is related to research published as "Botz-4-Sale: Surviving Organized DDoS Attacks That Mimic Flash Crowds," *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI '05)* (Berkeley: USENIX Association, 2005).

CONSIDER THE FOLLOWING SCENARIO: Alyssa Hacker subverts tens of thousands of machines by using a worm and then uses these zombies to mount a distributed denial of service attack on a Web server. Alyssa's zombies do not launch a SYN flood or issue dummy packets that will only congest the Web server's access link. Instead, the zombies fetch files or query search engine databases at the Web server. From the Web server's perspective, these zombie requests look exactly like legitimate requests, so the server ends up spending a lot of its time serving the zombies, causing legitimate users to be denied service.

Such an attack, which we call CyberSlam, is disconcertingly real. In a recent FBI case, a Massachusetts businessman hired professionals to DDoS his competitor's Web site [1]. Like any other online business, the competitor had a search engine back end. So the professionals used a large botnet to flood the competitor's site with a massive number of queries, bringing it down for almost a week. Several extortion attempts at online gaming and gambling sites used similar attacks [2].

Why CyberSlam? If you think about it, there are some real reasons why CyberSlam attacks happen. First, we know that many large botnets exist. Zombie machines are typically compromised by worms, viruses, or malware, and the zombies are controlled by remote botnet controllers over IRC channels [3, 4, 5]. Second, there is a great incentive to mimic the browsing patterns of legitimate users. It avoids detection by standard filters and intrusion detection boxes that routinely identify and block anomalous traffic. This is especially important for organized DDoS mafia, because for them the botnet is a *reusable* resource that they would like to protect. Finally, in CyberSlam an attacker is doing little while the server does a lot. By sending a single HTTP packet containing a small request, the attacker can make the server reserve sockets, TCP buffers, and an application process, and do significant database processing or congest some other server bottleneck.

So how can a system administrator deal with CyberSlam attacks? Let us look at some existing techniques. First, how about using passwords for authentication? Passwords don't exist for most Web sites (e.g., Google), because they are cumbersome to manage both for the site and for the customers. More im-



Our website is experiencing unusually high load.

To restrict automated access we require code verification.

Please enter the code shown in the image below:

FIGURE 1: GRAPHICAL PUZZLE

portant, to check a password the server has to establish a TCP connection with the client, reserve a socket and a server worker process, and search the password database, so an attacker can simply DDoS the password-checking mechanism. Second, computational puzzles make the client do some heavy computation before giving them service. But computation is typically abundant in a botnet, and solving a puzzle for every request slows down all the normal users. So, we need a new approach to counter CyberSlam attacks.

Here is a potential solution. Intuitively, online businesses care more about serving human users; so if the server can quickly identify the human users, it can serve their requests selectively and drop all the others. There are easy ways of distinguishing humans from zombies (e.g., the graphical puzzles used by Yahoo and Hotmail). Humans can solve these puzzles easily; zombies cannot do so at all. So when the server is overloaded, it sends a graphical puzzle, as shown in Fig. 1, to everybody and serves only those who answer correctly.

Challenges

Are we done, then? Unfortunately, the answer is no. There are three main challenges with using graphical puzzles. First, in a typical setup, sending the graphical puzzle allows an unauthenticated client to establish a TCP connection, hog sockets, TCP buffers, and application processes, and force context-switches from the kernel network stack up into application space, so attackers can easily DDoS this authentication mechanism. Second, graphical puzzles have a bias against users who cannot (disabled users) or will not (due to inconvenience) solve the puzzle. By forcing a server to use graphical puzzles, the attacker has already won—she denies access to all such users. Third, a server has to divide its resources between authenticating new users and serving the users it has already authenticated. This is a tricky problem. If the server authenticates every new user, it may run out of resources to serve users who are already authenticated, leading to unnecessary starvation. If, on the other hand, the server authenticates too few new arrivals, then it may not have enough users to serve and may go idle.

Introducing Kill-Bots

We present Kill-Bots, a simple, cheap, and effective software modification to the server's operating system that distinguishes friend from foe. The core principle is simple: do not allow clients to reserve any server resource until they are authenticated. Kill-Bots kicks in whenever a Web site is in danger of being overwhelmed by requests. The software asks requestors to solve a simple graphical puzzle before granting access to server resources such as buffer space. Once a client solves the graphical puzzle, she is given an HTTP cookie so that she can obtain service for some time without having to solve another puzzle. Addresses that repeatedly request access to the server without solving the puzzle are blacklisted automatically. When the load on the Web server decreases, it stops issuing puzzles and accepts requests from non-blacklisted addresses, so even real users who did not solve the puzzle gain access. Finally, Kill-Bots efficiently divides server resources by adapting the probability with which new users are authenticated. A Kill-Bots server neither accepts more users than it can serve nor goes idle by not authenticating enough new users. Fig. 2 shows how these individual pieces fit together in Kill-Bots, and Fig. 3 shows the HTML for the puzzle.

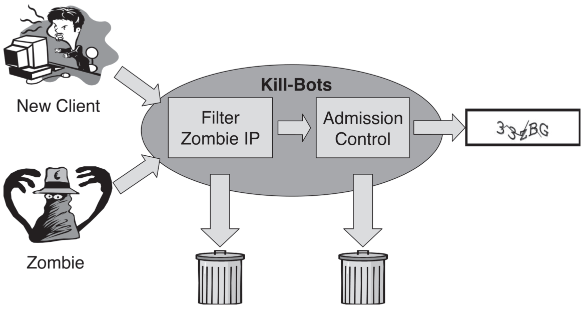
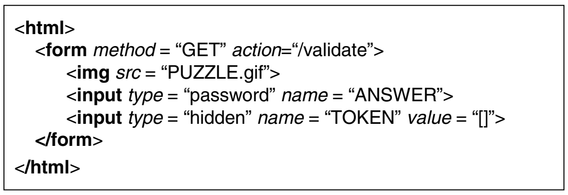


FIGURE 2: HANDLING ZOMBIES WITH KILL-BOTS



```
<html>
  <form method = "GET" action="/validate">
     <img src = "PUZZLE.gif">
     <input type = "password" name = "ANSWER">
     <input type = "hidden" name = "TOKEN" value = "[]">
  </form>
</html>
```

FIGURE 3: HTML SOURCE FOR THE PUZZLE

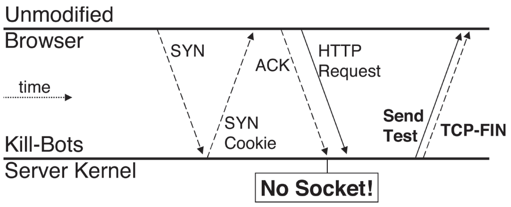


FIGURE 4: KILL-BOTS MODIFIES SERVER'S TCP STACK TO SEND TESTS TO NEW CLIENTS WITHOUT ALLOCATING A SOCKET OR OTHER CONNECTION RESOURCES

Let us briefly look at each of the main components of Kill-Bots. The modified network stack is shown in Fig. 4. When, the client sends a TCP SYN, Kill-Bots responds with a SYN cookie. The SYN cookie is a standard defense mechanism that serves two purposes. First, it filters requests from clients with spoofed IP addresses. Second, it allows the server to continue the TCP handshake without maintaining any state for the half-open connection. Upon receiving the SYN cookie, the client responds with an acknowledgment completing the TCP handshake. At this point, the standard network stack allocates a socket, reserves TCP buffers, and passes the request onto an application-space process. All of this is done for every attacker request and is quite costly. More important, these resources remain allocated until the client responds with a FIN. An attacker can simply hog resources by not sending the FIN. To avoid this, Kill-Bots ignores the acknowledgment. Instead, it looks at the first data packet that contains the same acknowledgment number and peeps into the HTTP header to confirm that this is a previously authenticated client with a valid HTTP cookie. If not, the modified kernel immediately sends the graphical puzzle and a TCP FIN without creating a socket or reserving any resources, as shown in Fig. 4.

Recall the second challenge: Graphical puzzles have a bias against users who cannot or will not answer the puzzle. So whenever an attacker forces the server to use graphical puzzles, these users are denied access. Kill-Bots uses the following observation of client behavior: a human user who does not answer the graphical puzzle will only retry a couple of times to see if he can access the server without answering; attackers, on the other hand, will have to continuously bombard the server with requests and fetch the graphical puzzles or the attack goes away. There is a simple difference between the human users and attackers now; the attackers have many more *unanswered* puzzles than the normal users. Kill-Bots uses a bloom filter to track the number of unanswered puzzles per IP and drops all requests from an IP if its associated bloom counters cross a limit, say, 32.

Recall the third challenge of dividing resources between serving authenticated users and authenticating new users. Kill-Bots deals with this by probabilistically authenticating new users and dropping others. The authentication probability adapts to server conditions. Intuitively, whenever the server is lightly loaded, the authentication probability increases so that Kill-Bots authenticates more new users, and when the server is heavily loaded it decreases. I will defer the specific details of the adaptation, but the tricky parts are *how much can one increase the authentication probability without overloading the server* and *how long it takes to adapt to changing conditions*. Note that small increments would reduce the chance of overshooting, while large increments would react to changing conditions and get you to the optimal operating condition quicker. We reconcile these contradictory preferences by using techniques from control theory.

In experiments, a Kill-Bots-protected Web server successfully endured five times as many hits as an unprotected Web server could tolerate. Not only did the Web server stay online, but protected Web sites also maintained speedy response times, even during the height of the attacks. You might wonder what would happen during a flash crowd, i.e., when the server overload is caused by a large number of legitimate requests. Kill-Bots improves both server hit-rate and response times by using the adaptive authentication probability mechanism to quickly drop new users who cannot be served. This ensures that server resources are not wasted on requests that are going to be dropped at a later time.

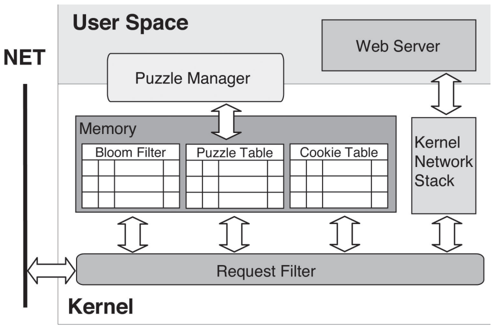


FIGURE 5: A MODULAR REPRESENTATION OF THE KILL-BOTS CODE

Using Kill-Bots

From a practical standpoint, here is how you could use Kill-Bots to protect your own Web server. Fig. 5 is a modular representation of the Web server using Kill-Bots. Kill-Bots doesn't require an extra server; it is a software patch to the operating system of the server kernel. Kill-Bots needs a store of graphical puzzles. Generating the graphical puzzles is relatively easy, for example using the JCAPTCHA software [6], and can done on the server itself during periods of relative inactivity or on a different dedicated machine. Also, puzzles may be purchased from a trusted third party. Kill-Bots has modest overhead; it uses 10MB of RAM to cache graphical puzzles, maintain the bloom filter that blacklists zombie

IPs, and maintain state for each cookie. A kernel thread periodically loads fresh graphical puzzles into memory. The per-request overhead is also quite small. On a 2.4GHz PIV workstation, peeping into HTTP requests costs 8 microseconds of server time, and serving a puzzle costs 31 microseconds.

Why Does Kill-Bots Matter?

Worries over distributed denial-of-service attacks are spreading. It is depressing, yet true, that the future will see many more organized DDoS attacks. Most Web server defenses use authentication procedures that are easily outwitted and require huge excesses in the form of replicated content, multiple CPUs, fancy hardware, and extra bandwidth. Kill-Bots is much cheaper and can be deployed easily; it requires no changes in users' Web browsers and works with the very large number of Web servers running Linux. Although Kill-Bots occasionally misclassifies legitimate users as zombies, it allows Web sites under attack to remain available and so keeps the Web open for business, while barring the way to thieves and vandals.

REFERENCES

[1] K. Poulsen, "FBI Busts Alleged DDoS Mafia," 2004: http://www.securityfocus.com/news/9411.

[2] J. Leyden, "East European Gangs in Online Protection Racket," 2003: http://www.theregister.co.uk/2003/11/12/east_european_gangs_in_online/.

[3] J. Leyden, "The Illicit trade in Compromised PCs," 2004: http://www.theregister.co.uk/2004/04/30/spam_biz/.

[4] E. Hellweg, "When Bot Nets Attack," *MIT Technology Review,* September 2004.

[5] L. Taylor, "Botnets and Botherds": http://sfbay-infragard.org.

[6] JCAPTCHA: http://jcaptcha.sourceforge.net/.