The Case for Public Work

Wu-chang Feng Ed Kaiser
Portland State University
{wuchang,edkaiser}@cs.pdx.edu

***Abstract*—Whether it is port scans, spam, or distributed denial-of-service attacks from botnets, unwanted traffic is a fundamental problem in all networked systems. Although proof-of-work has been proposed as a mechanism for thwarting such attacks, few proof-of-work systems have been successfully deployed. One of the problems in the proof-of-work approach is that the systems that issue and verify puzzles are typically located at or near the server edge. Rather than eliminate the denial-of-service problem, such approaches merely shift the problem from the service itself to the proof-of-work systems protecting the service. As a result, adversaries can disable services by flooding the issuer, by flooding the verifier, or by flooding all of the network links that lead to the issuer and verifier.**

To address this problem, this paper proposes a new approach for building proof-of-work systems based on publicly verifiable client puzzles. The system works by issuing a single "public work function" that clients must solve for each of its subsequent requests. Because the work function is publicly verifiable, any network device at the client's edge can verify that subsequent traffic will be accepted by the service. The system mitigates floods to the issuer since only a single work function needs to be given per client, thus allowing duplicate requests and replies to be supressed. The system mitigates floods to the verifier and across links leading to the server edge by allowing the verifier to be placed arbitrarily close to the client adversary.

I. Introduction

With the continued presence of spam, scans, and botnets on the Internet, it is clear that unwanted traffic still poses significant challenges. One of the problems is the lack of mechanisms for controlling who or how someone accesses public services. Once its location is known, unsolicited traffic can immediately reach any service. There have been a number of approaches for combating unwanted traffic with solutions ranging from indirection [1], [2], [3], [4], filtering [5], [6], [7], capabilities [8], [9], and proof-of-work [10], [11], [12], [13], [14], [15], [16].

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0627752.

The above systems have many salient features that must be incorporated to adequately address the problem of unwanted traffic. Indirection provides the ability to hide or dynamically relocate a public service in order to prevent malicious clients from reaching the service indefinitely. Filtering is necessary to stop unwanted traffic as close to the source as possible. Capabilities are necessary to give services dynamic, fine-grained control at the request level over access. Finally, proof-of-work is necessary to ensure adversaries commit as many resources as they are requesting from a service.

This paper proposes an approach that integrates aspects of each of the techniques into a mechanism based on "public work". The crux of the scheme is simple. The service, as part of advertising its location, provides a publicly verifiable work function that the client must solve in order to correctly reach the service. A client must attach a valid answer to this function along with its service request. If the client does not, then any network device that has recorded the previous advertisement can verify that subsequent requests are not wanted by the service and can then drop them long before they reach their destination. Since the service itself controls the difficulty of the work function passed back to the clients, it can control its reachability at a fine granularity.

II. Public Work Approach

A. Basic approach

Figure 1 shows the basic approach. In the scheme, as part of advertising its location, a service also supplies a source-specific work function whose solution must be calculated by the client and attached on subsequent service requests before being given service. The work function can be delivered either on-demand (e.g. piggy-backed on DNS replies) or a-priori (e.g. via key insertions into a DHT [3], periodic publishing [17], or prefetching [18]). The novel property of the work function is that it is publicly verifiable, that is, any network device that receives the advertisement can determine the validity of subsequent service requests. In addition, while the

978-1-4244-1697-4/07/$25.00 ©2007 IEEE

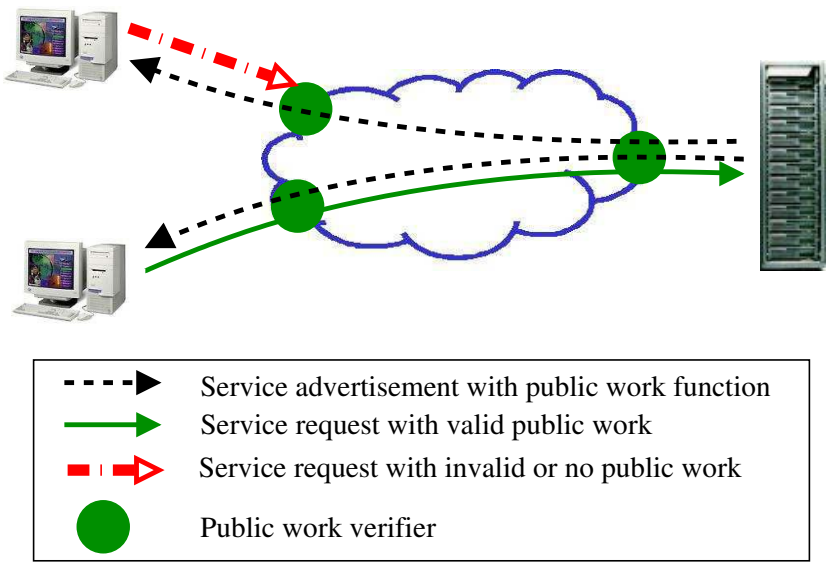

Fig. 1. The public work approach.

work function is easy to generate and verify, it requires resources to calculate a correct answer. This property enables devices at the client's edge to detect and filter out traffic that is unwanted by Internet services. As shown in the figure, requests that do not have valid public work attached are filtered as soon as they reach a network device that verifies public work.

The public work scheme draws from indirection in enabling the service to avoid targeted attack by allowing it to dynamically change its "reachable" locations via continuous updates to its public work function. In order to construct a valid request that will reach a service, a client must have a correct answer to a recent public work function along with the service location. Furthermore, if the function is source-specific, the service can control these "reachable" locations on a per-client basis.

The scheme draws on aspects of filtering by supporting destination-controlled filtering at the edges of the network. Specifically, filters or verification points can be placed at any point in the network that receives both the public work advertisement and the subsequent request. For example, the first-hop router at the client could store the public work advertisements and then check that subsequent requests satisfy it. A service that is receiving unwanted traffic from particular sources could advertise to those sources a public work function with a high degree of difficulty or one without a solution. If the source chooses to solve the function, it is slowed considerably. If it ignores the function, then due the public nature of the advertisement, intermediate network nodes can drop subsequent requests.

The scheme draws on capabilities by giving the service complete control over the clients that access it since the work function can be made specific to the requesting client. The difficulty of the function directly controls how clients are given access. The public nature of the verification allows nodes at the edges of the network to validate subsequent capabilities.

The scheme also embodies ideas of proof-of-work. The work function given to the client is a puzzle of a certain difficulty. The client must solve the puzzle correctly and attach the answer to a subsequent request in order to reach the service. The key difference of the scheme is that the solution to the public work function or puzzle is publicly verifiable.

B. Sudoku: A Toy Example

As a toy example of a public work function, consider a system where a service issues an n-digit "Sudoku" puzzle to each client along with its location. Based on the amount of resources the service wishes the client to consume, n is either increased or decreased. Given the puzzle, clients must then attach a valid solution before its requests are forwarded. As Sudoku is an NP-complete problem [19], it is difficult for clients to calculate solutions while it is easy to check valid solutions assuming the verifier has previously recorded the puzzle. While Sudoku puzzles are an interesting example, they have several drawbacks that make them unsuitable for use in the network. One drawback is that it is unclear how to efficiently generate appropriate puzzles with configurable

978-1-4244-1697-4/07/$25.00 ©2007 IEEE

difficulty. Another is that puzzle answers could be reused amongst clients leading to the potential for off-line storage and pre-computation attacks.

C. Goals for a Public Work Function

To develop an appropriate public network puzzle, we first examine ideal goals for one. Formally, a public work function is any computational puzzle whose answer can be publicly verified. While many public work functions might exist, the following four properties are necessary for a public work function to be practical in this context.

- Fast issuing: Generating the function must add minimal overhead to the service advertisement process.
- Fast verification: While finding a solution to the function must be non-trivial, verifying a solution must add minimal overhead to request forwarding.
- Flexible binding: The function must be flexible enough to bind to various scales of communication, such as packets, flows, or flow aggregates.
- Limited precomputation and replay: The function must resist precomputation and replay attacks.

D. A Novel Public Work Function

To meet the above requirements, we propose a new class of public work functions called *targeted cryptographic hash function reversal*. In its simplest form, the service attaches a per-client random number N_C and difficulty D_C with its service advertisement. The client must then calculate an answer A such that:

$$SHA1(A, F, N_C) \equiv 0 \bmod D_C$$

where F represents flow properties of the subsequent request such as the addresses and ports of the source and destination. In its use here, SHA1 is assumed to be a one-way function that has uniformly distributed random output. It must be one-way so that finding the unknown parameter A involves a non-trivial search. It must also have uniformly random output so that the solver is expected to try D_C distinct values for A before finding a value that satisfies the equation. Note that the function itself changes on a per-request basis with F, forcing distinct client requests to calculate new answers. While a client might simply reuse F on subsequent requests, such duplicates are easy to identify and drop at the client edge using techniques such as Bloom filters.

In this construction, the solver must back-out the message that produces a given digest, commonly referred to as a preimage attack. The construction thus exploits the hash function's *preimage-resistant* property.

IV size	Input size	Hash effort
20 bytes	64 bytes	1144 cycles

TABLE I
SHA1 HASH COMPRESSION FUNCTION SPEEDS.

The message to be hashed is created from the concatenation of the puzzle parameters (the answer, the flow, and the random number). Since the input size of most hash functions is large enough to incorporate all of the parameters, the construction can be completed in a single execution of the hash's internal compression function, thus making answer verification extremely fast. Although hash functions have not been proven to have uniformly distributed output, experimental evidence indicates that many of them do [20].

The function meets each of the requirements:

- Fast issuing: The issuer generates a single random number N_C and a difficulty D_C to issue a new work function to a particular client. The random number N_C is periodically updated to maintain freshness. The per-client difficulty D_C can be obtained via a simple table lookup or a counting Bloom filter [21].
- Fast verification: The verifier only needs to perform a table lookup to retrieve the appropriate work function (N_C, D_C) and a single SHA1 hash to check the answer A in order to determine the validity of the subsequent communication. Table I shows the number of clock cycles on a 1.8GHz Pentium 4 system required to execute the SHA1 hash function in the construction above. The overhead of the hash is around 1100 cycles ($< 1us$).
- Flexible binding: The parameters that define F are configurable and can include the source and destination addresses, ports, and other protocol fields.
- Limited precomputation and replay: The validity of a work function is directly controlled by the service. Precomputation and replay are limited since the work function is periodically updated by generating and advertising a new random number N_C and difficulty D_C.

III. Towards Practical Public Work Systems

While the public work mechanism is promising, there are a significant number of research problems that must be overcome in order to build real systems that can leverage the approach to effectively reduce unwanted traffic.

978-1-4244-1697-4/07/$25.00 ©2007 IEEE

A. The granularity problem

One of the key issues that must be considered is *what* to protect with public work. Because the parameter F in the work function can use any number of properties in the subsequent request, the mechanism is flexible and can accommodate numerous approaches. For example, one could use public work functions to protect specific content by attaching them to items such as URIs, to content advertisements in peer-to-peer networks like Freenet [22], or to keys in DHTs [2], [3], [23], [24]. Another way public work could be used is to protect TCP connection setups by attaching work functions to the TCP handshake [14], [25]. Finally, public work could be used to protect the location of services themselves by attaching them to beacons in SOS and Mayday [1], [4] or to DNS [26].

B. The issuing problem

Another fundamental problem in building public work systems is how to protect the issuer itself against denial-of-service. The public work approach provides two salient features that are useful against such attacks. The first is that issuing a public work function consists of advertising a per-client random number N_C and difficulty D_C. This makes the issuing mechanism rather trivial and makes implementations of the public work issuer efficient and difficult to overwhelm. The second is that the client only needs to be given a single, up-to-date public work function from the issuer. The function, while only given once, must then be solved for each new request for service by the client. Because a legitimate client only needs to be issued a single public work function, a large number of duplicate requests to the issuer from a malicious client can be easily identified and dropped at the network edge. An adversary attempting to disable the issuer using a botnet can only do so using a single request per compromised machine. Such a restriction requires the adversary to compromise an enormous number of machines in order to sustain an attack against the issuer. Still, for any system using the public work approach, it is important to understand the number of machines an adversary must compromise in order to completely shut down access to the issuer by legitimate clients.

C. The delivery problem

Related to the issuing problem is how public work functions are delivered to clients. There are a range of options that could be considered based on what is being protected. The public work function could be embedded in URIs or HTTP headers, attached to TCP SYN/ACKs [25] or TCP puzzles [14], or included in DNS advertisements. It could also be delivered via a completely separate protocol or via new ICMP protocol messages. The choice of delivery impacts the ability to handle floods and spoofing. For example, consider a service that embeds a public work function within TCP SYN/ACKs. An adversary might flood the service with spoofed TCP SYN packets from a target victim. Assuming the issuer ignores duplicate requests for public work functions, when the target victim later attempts to get its associated work function, its request will be dropped. Such a problem does not exist when considering delivery mechanisms that are preceded by a three-way handshake. Any public work system must include a delivery mechanism that ensures that legitimate clients are always able to obtain their corresponding work function.

D. The verification problem

Much like floods against the issuer, public work systems must be able to thwart denial-of-service attacks against the verifier. The core contribution of the public work approach is the ability to have any network device, and in particular, those devices close to the adversary, perform the verification of work. In addition, because verification consists of executing the work function $SHA1$ with the attached values of F, N_C, and the answer A, the verification process can be made extremely efficient. In this case, the verifier only needs to look up the previously recorded public work function (N_C, D_C) and ensure that:

$$SHA1(A,F,N_C) \equiv 0 \bmod D_C$$

Assuming F is the flow identifier of the subsequent request, this requires only a single pass through the SHA1 compression function. Such an operation can be performed in under 1μs on a commodity PC platform.

E. The asymmetry problem

While verification is always performed at the server edge, one of the key challenges in the public work approach is *where* to place verifiers at the client edge. The placement of client-side verifiers is driven by the fact that the verifier must see *both* the public work function and the subsequent request. Due to the inherent asymmetry in routing in today's Internet and the desire to drop unwanted traffic as close to the source as possible, such verification would ideally be performed by the client operating system itself. However, since an adversary can

978-1-4244-1697-4/07/$25.00 ©2007 IEEE

disable and modify software running on the client, this approach can be subverted. Another approach would be to embed the verifier in first-hop routers or at client-side ingress filters [27]. While such an approach would be difficult for adversaries to subvert, it requires state to be kept in the network in the form of per-client public work functions. An approach that combines the best of both worlds is to use tamper-resistant code embedded in hardware at the client. Such a facility is supported by Intel's Active Management Technology (AMT) platform which is included in most modern Intel processors [28]. The AMT platform consists of a separate, secure co-processor that only runs code signed by Intel. The co-processor is hidden from both the user and the operating system and is currently being used to filter outgoing traffic that has been determined to be malicious and to securely perform integrity checks on critical software.

F. The difficulty problem

As described earlier, the impact that proof-of-work systems have on innocent clients often causes them to fail. From an economic standpoint, in order for a proof-of-work scheme to be effective, the amount of work required of the "good guys" and the amount of work required of the "bad guys" must differ significantly [29]. In particular, due to the sheer number of compromised systems that exist today, a global difficulty setting is easily overcome by brute force. Stated slightly differently, no proof-of-work system can function properly unless difficulties are properly tailored based on the history of client usage. With that in mind, one of the design requirements for public work systems is that there is a mechanism for the server or issuer to continuously keep track of per-client resource consumption and to deliver public work based on this resource consumption. The accounting mechanism must itself be efficient and handle large numbers of potential sources [30]. Rough estimates of the current sizes of "Botnets" indicate that large networks have up to 100,000 hosts, although this figure now appears to be decreasing [31]. In addition, the accounting mechanism must keep track of usage over long time scales in order to thwart both persistent attacks as well as shrews [32]. It is an open question whether or not one can effectively manage the difficulty of public work functions to thwart all forms of denial-of-service attacks. Effective algorithms for properly managing per-client difficulties are essential in order to successfully deploy systems based on public work and to create long-term incentives for proper behavior in networks.

G. The replay problem

One problem unique to puzzle systems is the ability for an adversary to replay previously calculated answers indefinitely. In the public work system, there are several ways to address this problem. One would be to immediately replace the client's current work function with a much more difficult one whenever a server detects a client reusing an answer. Another would be to rely on the verifier at the client edge to detect when individual answers are being reused and to transparently drop such requests. While both ways effectively shut down the attack, they do so at the expense of the issuer and verifier. Any public work system must address replay attacks and develop mechanisms for ensuring that long-term replay of answers is not beneficial to the adversary.

H. The spoofing problem

Although the proliferation of ingress filtering [27], [33], [34] has made spoofing attacks rare, any public work system must be able to handle spoofing without causing service disruption to legitimate clients. There are several potential ways an adversary can use spoofing to attack public work systems. As described earlier, the adversary could spoof requests for work functions from a large number of clients to a single issuer to either keep those clients from obtaining a work function or to disable the issuer altogether. The adversary could employ a reflector attack [35] and spoof requests for work functions from a target victim to a large number of servers to flood the target victim and prevent it from obtaining any work functions. The adversary could employ a poisoning attack [36] and spoof the work function itself to keep a targeted victim from accessing a particular service by giving it an extremely difficult function to solve. The adversary could capture the public work function of a target victim, solve the function, and spoof large numbers of requests from the victim in order to drive up the victim's difficulty. Finally, the adversary could spoof large amounts of traffic with bogus answers in an attempt to disable the verifier. There are several approaches for tackling the spoofing problem based on what is being spoofed. For example, connection spoofing on the Internet is typically handled using sufficiently random sequence numbers and a 3-way handshake [37], while DNS and web-site spoofing is handled using public-key cryptography via DNSsec and TLS [38]. Appropriate mechanisms must be developed in order to make public work systems resilient to spoofing attacks.

978-1-4244-1697-4/07/$25.00 ©2007 IEEE

IV. Conclusion

The public work function approach adds two significant new features to current proof-of-work systems. To mitigate denial-of-service attacks against the issuer, its work functions are given on a per-client basis instead of a per-request basis. To mitigate floods against the verifier and the network links leading to the server, it supports public verification of work that enables network devices close to the client to drop unwanted traffic. While there are many fundamental problems that still need to be addressed before it can be used, public work is a promising approach for combating the problem of unwanted traffic in today's Internet as well as in a "clean-slate" network design.

References

[1] A. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," in *ACM SIGCOMM*, August 2002.

[2] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet Indirection Infrastructure," in *ACM SIGCOMM*, August 2002.

[3] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica, "Taming IP Packet Flooding Attacks," in *HotNets*, November 2003.

[4] D. Andersen, "Mayday: Distributed Filtering for Internet Services," in *USENIX Symposium on Internet Technologies and Systems*, March 2003.

[5] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling High Bandwidth Aggregates in the Network," *ACM SIGCOMM CCR*, vol. 32, no. 3, July 2002.

[6] M. Handley and A. Greenhalgh, "Steps Toward a DoS-resistant Internet Architecture," in *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, August 2004.

[7] D. Yau, J. Lui, and F. Liang, "Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles," in *Proceedings of IWQoS*, May 2002.

[8] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet Denial-of-Service with Capabilities," in *HotNets*, November 2003.

[9] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting Network Architecture," in *ACM SIGCOMM*, August 2005.

[10] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," in *CRYPTO*, August 1992.

[11] A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Defense Against Connection Depletion," in *NDSS*, February 1999.

[12] T. Aura, P. Nikander, and J. Leiwo, "DoS-Resistant Authentication with Client Puzzles," in *Workshop on Security Protocols*, April 2000.

[13] A. Back, "Hashcash: A Denial of Service Counter-Measure," Tech. Rep., Cypherspace, August 2002, http://www.hashcash.org/papers/hashcash.pdf.

[14] X. Wang and M. Reiter, "Defending Against Denial-of-Service Attacks with Puzzle Auctions," in *IEEE Symposium on Security and Privacy (S&P)*, May 2003.

[15] W. Feng, "The Case for TCP/IP Puzzles," in *ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA)*, August 2003.

[16] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Host identity protocol," February 2007, Internet Draft draft-ietf-hip-base-07.txt.

[17] M. Handley and A. Greenhalgh, "The Case for Pushing DNS," in *HotNets*, November 2005.

[18] V. Ramasubramanian and E. Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," in *ACM SIGCOMM*, August 2004.

[19] T. Yato, "Complexity and Completeness of Finding Another Solution and Its Application to Puzzles," Tech. Rep., University of Tokyo Master Thesis, January 2003.

[20] M. Bellare and T. Kohno, "Hash Function Balance and its Impact on Birthday Attacks," in *EUROCRYPT*, May 2004.

[21] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," in *ACM SIGCOMM*, September 1998.

[22] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Lecture Notes in Computer Science*, vol. 2009, 2001.

[23] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *ACM SIGCOMM*, August 2001.

[24] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *ACM SIGCOMM*, August 2001.

[25] D. Bernstein, "SYN Cookies," http://cr.yp.to/syncookies.html.

[26] P. Mockapetris, "Domain Names: Concepts and Facilities," November 1983, RFC 882.

[27] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing," *RFC 2827*, May 2000.

[28] "Intel Active Management Technology," http://www.intel.com/technology/manage/iamt/.

[29] B. Laurie and R. Clayton, "'Proof-of-Work' Proves Not to Work'," in *Workshop on Economics and Information Security*, May 2004.

[30] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," in *ACM SIGCOMM*, August 2002.

[31] D. Kawamoto, "Bots Slim Down to Get Tough," November 2005, http://news.com.com/2102-7355_3-5956143.html.

[32] A. Kuzmanovic and E. Knightly, "Low-rate TCP-targeted Denial of Service Attacks (the Shrew vs. the Mice and Elephants)," in *ACM SIGCOMM*, August 2003.

[33] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," March 2004.

[34] R. Beverly and S. Bauer, "The Spoofer Project: Inferring the Extent of Internet Source Address Filtering on the Internet," in *SRUTI*, July 2005.

[35] V. Paxson, "An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks," *ACM SIGCOMM CCR*, vol. 31, no. 3, July 2001.

[36] K. Huagsness, "DNS Cache Poisoning Detailed Analysis Report, Version 2," March 2005, http://isc.sans.org/.

[37] M. Zalewski, "Strange Attractors and TCP/IP Sequence Number Analysis," Tech. Rep., Bindview, April 2001, http://razor.bindview.com/publish/papers/tcpseq.html.

[38] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," March 2005.

978-1-4244-1697-4/07/$25.00 ©2007 IEEE