

Forum:[Hack Forums](#)

Thread: [\[Challenge\] Client puzzle for denial-of-service.](#)

Phizo - August 12, 2017, 2:58 pm

The context isn't necessary for solving the challenge. It is just there for anyone who is interested in what application this challenge is involved in.

Context

Here is some context based on a protocol based on this idea:

Quote:Client Puzzle Protocol (CPP) is a computer algorithm for use in Internet communication, whose goal is to make abuse of server resources infeasible. It is an implementation of a proof-of-work system (POW).

The idea of the CPP is to require all clients connecting to a server to correctly solve a mathematical puzzle before establishing a connection, if the server is under attack. After solving the puzzle, the client would return the solution to the server, which the server would quickly verify, or reject and drop the connection. The puzzle is made simple and easily solvable but requires at least a minimal amount of computation on the client side. Legitimate users would experience just a negligible computational cost, but abuse would be deterred: those clients that try to simultaneously establish a large number of connections would be unable to do so because of the computational cost (time delay). This method holds promise in fighting some types of spam as well as other attacks like Denial of Service.

Here is an example of how a client and server typically interact based on this protocol (or a similar protocol):

Client sends server a hello message.

The server:

Determines k and generates N_s and T_s .

Sends $\{T_s, k, N_s\}$ to the client.

The client:

Generates N_c .

Verifies T_s .

Finds X such that $H(C, N_s, N_c, X) = 111...111Y$.

Sends $\{S, C, N_s, N_c, X\}$ to the server.

The server:

Verifies that N_s is recent.

Checks if $\{C, N_s, N_c\}$ has been used before.

Checks if $H(C, N_s, N_c, X) = 111...111Y$.

If it does, the resource is committed.

$\{C, N_s, N_c\}$ is stored.

$\{S, C, N_c\}$ is sent to the client.

The operation can now continue.

Notation: H : A hash function.

C : Client ID.

N_s : Server nonce.

N_c : Client nonce.

T_s : Timestamp.

k : Puzzle difficulty level.

Y : The rest of hash value.

S : The resource to be submitted.

Goal

Given C , N_s , N_c , use brute-force to find X , the set of printable characters, such that $H(C, N_s, N_c, X)$ contains has the k leftmost bits set: $H(C, N_s, N_c, X) = 111...111Y$, where Y can be any value. Let $k = 24$ and let H represent the cryptographic hash algorithm MD5. The hash algorithm does not need to be secure for this.

Now, the parameters for the hash function. Let C be a random 7-digit unsigned integer, $N_s = "1337XYZ"$, $N_c = "HackForums"$, and make X a 20-byte string. C being random is to make each individual's solution unique.

The maximum number of guesses is 2^k , and the average is $2^k / 2 = 2^{(k-1)}$. In general, for n sub-puzzles, the complexity is $n * 2^{(k-1)}$ on average. I have made $n = 1$ and k reasonable to make this challenge not too computationally difficult. Just enough to make it so you have to know what you're doing in order to solve it. Note that you should not be permuting all bits of X . You should permute 2^k at most. Though the best way to go about it is to take a random subsets and test if any of its permutations are solutions.

Good luck.

WOLGA - August 13, 2017, 10:26 am

Are you sure that the maximum number of guesses for a given difficulty k is equal to 2^k ?

We can imagine a set of 16777216 (2^{24}) guesses, of which none starts with 0, much less 24.

Phizo - August 13, 2017, 10:29 am

I'm not sure what you're trying to say, but yes, I am sure. For k bits, there are 2^k possible values.

WOLGA - August 13, 2017, 10:31 am

(Aug 13, 2017 10:29 AM)Phizo Wrote: [\[CITE\]](#) I'm not sure what you're trying to say, but yes, I am sure. For k bits, there are 2^k possible values.

But for MD5 (128 bits), it is not certain that each guess will result in a distinct first 24 bits.

Phizo - August 13, 2017, 10:47 am

(Aug 13, 2017 10:31 AM)WOLGA Wrote: [\[CITE\]](#) But for MD5 (128 bits), it is not certain that each guess will result in a distinct first 24 bits.

Collision is besides the point here. It is actually helpful considering this is brute-force. The codomain of a 128-bit hash function which outputs a hexadecimal number/string is the set $[0x0000000000000000, 0xFFFFFFFFFFFFFFFF]$. The maximum possible value is $2^{128} - 1$ unsigned or $2^{127} - 1$ signed. All the bits may be set. So, yes, I'm sure for k leftmost bits, there are 2^k possible values.

WOLGA - August 13, 2017, 10:51 am

(Aug 13, 2017 10:47 AM)Phizo Wrote: [\[CITE\]](#) Collision is besides the point here. It is actually helpful considering this is brute-force. The codomain of a 128-bit hash function which outputs a hexadecimal number/string is the set $[0x0000000000000000, 0xFFFFFFFFFFFFFFFF]$. The maximum possible value is $2^{128} - 1$ unsigned or $2^{127} - 1$ signed. All the bits may be set. So, yes, I'm sure for k leftmost bits, there are 2^k possible values.

I'm not talking about collision of the whole hash.

You're right, that for the first k bits, there are 2^k possibilities, but a set of 2^k guesses won't necessarily permute them all, for example, it may change the other $128-k$ bits, but leave the first k unchanged.

Phizo - August 13, 2017, 11:03 am

(Aug 13, 2017 10:51 AM)WOLGA Wrote: [\[CITE\]](#) I'm not talking about collision of the whole hash.

You're right, that for the first k bits, there are 2^k possibilities, but a set of 2^k guesses won't necessarily permute them all, for example, it may change the other $128-k$ bits, but leave the first k unchanged.

That is precisely the point of the challenge. You are to find the hash value such that the k leftmost bits are unset. You are reciting me my own challenge. I don't understand why you think I don't get a challenge I've defined.

WOLGA - August 13, 2017, 11:18 am

(Aug 13, 2017 11:03 AM)Phizo Wrote: [\[CITE\]](#) That is precisely the point of the challenge. You are to find the hash value such that the k leftmost bits are unset. You are reciting me my own challenge. I don't understand why you think I don't get a challenge I've defined.

Sorry, I'm pretty bad at explaining. I pretty much stumbled upon, by attempting this challenge, the fact that for a set of 2^{24} guesses, there does not need to be any one that has the first 24 bits zeroed.

I tried to cycle through all hashes of

Code "11363271337XYZHackForums"+XXX

Where XXX is any 3 byte (24 bit) sequence and I found that of all the possible sequences, none gave a md5 hash that started with 24 bits zeroed, contrary to your statement that 2^{24} guesses guarantees one such hash.

Edit: Sorry, it seems that there was an problem in the md5 library that I used, which messed this up: there should have been a byte sequence which gave a hash starting with 24 zeros, but my point that there is no guarantee stands.

Phizo - August 13, 2017, 11:43 am

(Aug 13, 2017 11:18 AM)WOLGA Wrote: [\[CITE\]](#) Sorry, I'm pretty bad at explaining. I pretty much stumbled upon, by attempting this challenge, the fact that for a set of 2^{24} guesses, there does not need to be any one that has the first 24 bits zeroed.

I tried to cycle through all hashes of

Code "11363271337XYZHackForums"+XXX

Where XXX is any 3 byte (24 bit) sequence and I found that of all the possible sequences, none gave a md5 hash that started with 24 bits zeroed, contrary to your statement that 2^{24} guesses guarantees one such hash.

You misunderstand. Just because there is a maximum of 2^k possible guesses doesn't mean that you should make your input consist of k bits. I will update the OP to say this, as I forgot. X should be 20 bytes in length. 2^k possible guesses means that you can solve this by permuting a maximum of 2^k bits of X . If you don't believe me, look up client puzzles. I assure you every resource you find will tell you that this is the case.

WOLGA - August 13, 2017, 3:00 pm

I am trying to show that your quote (below) is incorrect. I am trying to say that it can take more than 2^{24} guesses before finding one whose hash starts with 24 bits set to zero, i.e. 2^k guesses does not guarantee a hash with the first 24 bits set to zero

Quote:you can solve this by permuting a maximum of 2^k bits of X .

Since there are

Code $2^{(128-24)} = 2.028241e+31$ output values that do not contain start with 24 zeros, it follows that I can construct sets of 2^{24} guesses, none of which result in a hash with start with 24 zeros.

Phizo - August 13, 2017, 5:27 pm

It's not. Maybe you should do what I said and go read a paper on client puzzles, as all of them will tell you that it can be done in a maximum of 2^k guesses and $2^{(k-1)}$ on average. If all those papers disagree with what you're saying, then maybe it's the truth. I'm not trying to be rude. But you may see that while, yes, good hash functions are uniformly distributed, this is still not the case, the maximum number of guesses is truly 2^k by cross-referencing what I'm saying with papers on client puzzles.

WOLGA - August 13, 2017, 5:56 pm

(Aug 13, 2017 05:27 PM)Phizo Wrote: [\[CITE\]](#) It's not. Maybe you should do what I said and go read a paper on client puzzles, as all of them will tell you that it can be done in a maximum of 2^k guesses and $2^{(k-1)}$ on average. If all those papers disagree with what you're saying, then maybe it's the truth.

Let's assume that I have a 16777216-sided dice. If I roll this 16777216-sided dice 16777216 times, am I guaranteed to land any particular side?

You're trying to say something akin to "if you roll a 6-sided dice 6 times, you are guaranteed to land on 1 at least once".

Can you recommend one such paper which disagrees with me?

Phizo - August 14, 2017, 2:35 am

(Aug 13, 2017 05:56 PM)WOLGA Wrote: [\[CITE\]](#) Let's assume that I have a 16777216-sided dice. If I roll this 16777216-sided dice 16777216 times, am I guaranteed to land any particular side?

You're trying to say something akin to "if you roll a 6-sided dice 6 times, you are guaranteed to land on 1 at least once".

Can you recommend one such paper which disagrees with me?

That is not how cryptographic hash functions work. You are stepping into probability theory, which is not applicable here. Hash functions are not probabilistic. They're deterministic. Deterministic algorithms always produce the same output given a particular input. Defined more formally as a state machine, the machine will always go through the same sequence of states, with every current state determining the next state. From the perspective of the mathematical function which the deterministic algorithm computes, i.e. the hash function, if you iterate through all possible values of its domain as you invoke it, then you also iterate through all possible values in its codomain. The elements of the sets which are the domain and codomain do not change, nor does the mapping of these elements. Again, probability is not a factor here. You misunderstand how cryptographic hash functions work.

http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand12/Group2Mads/final/andreasGabrielsson_simonOsterman.final.pdf

Quote:When the server has generated and sent back P and t the client has to solve the puzzle. Since h is not invertible this has to be done by brute force [3]. The client simply tries a value v of length k and concatenate them with y , then it computes $h(v.y)$ and compare to z . If they match v is the solution to the sub-puzzle, again $.$ means bit-string concatenation. For each guess one hash value has to be computed and the number of guesses that has to be done depends on the value k . The maximum number of guesses is 2^k and the average is $2^k/2 = 2^{(k-1)}$, since P consists of n puzzles the average time to solve a full puzzle is $n \cdot 2^{(k-1)}$. Along with each value v for every sub-puzzle the solution that is sent back to the server consists of t and the message m that was sent to the server when asking for a puzzle

Here are some more:

http://discovery.csc.ncsu.edu/Courses/csc774-S07/T02.2.x_Client_puzzles.ppt

<https://www.google.com.au/patents/US7197639>

I also updated the puzzle solution to be such that k leftmost bits are set rather than not set, as that is what my solution does, and I can't be bothered changing parameters to compute another solution. It doesn't matter whether the bits are set or not. The complexity is identical. The puzzle is just to get a particular pattern of bits.

WOLGA - August 14, 2017, 7:23 pm

(Aug 14, 2017 02:35 AM)Phizo Wrote: [\[CITE\]](#) That is not how cryptographic hash functions work. You are stepping into probability theory, which is not applicable here. Hash functions are not probabilistic. They're deterministic. Deterministic algorithms always produce the same output given a particular input. Defined more formally as a state machine, the machine will always go through the same sequence of states, with every current state determining the next state. From the perspective of the mathematical function which the deterministic algorithm computes, i.e. the hash function, if you iterate through all possible values of its domain as you invoke it, then you also iterate through all possible values in its codomain. The elements of the sets which are the domain and codomain do not change, nor does the mapping of these elements. Again, probability is not a factor here. You misunderstand how cryptographic hash functions work. The deterministic quality of a hash function is not what is at question here.

Again, I am trying to say that under your implementation of the client puzzle, there is no guarantee that the client will solve the puzzle in 2^k guesses, since you're essentially rolling a 2^k sided dice on every guess.

(Aug 14, 2017 02:35 AM)Phizo Wrote: [\[CITE\]](#) I also updated the puzzle solution to be such that k leftmost bits are set rather than not set, as that is what my solution does, and I can't be bothered changing parameters to compute another solution. It doesn't matter whether the bits are set or not. The complexity is identical. The puzzle is just to get a particular pattern of bits. It doesn't matter what pattern of bits you're looking for. No particular pattern of 24 bits is guaranteed to show up in the hash of any of 2^{24} guesses.

(Aug 14, 2017 02:35 AM)Phizo Wrote: [\[CITE\]](#)

<http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand12/Group2Mads/final/andreasGabrielsson.simonOsterman.final.pdf>

Quote: When the server has generated and sent back P and t the client has to solve the puzzle. Since h is not invertible this has to be done by brute force [3]. The client simply tries a value v of length k and concatenate them with y , then it computes $h(v.y)$ and compare to z . If they match v is the solution to the sub-puzzle, again $.$ means bit-string concatenation. For each guess one hash value has to be computed and the number of guesses that has to be done depends on the value k . The maximum number of guesses is 2^k and the average is $2^k/2 = 2^{(k-1)}$, since P consists of n puzzles the average time to solve a full puzzle is $n \cdot 2^{(k-1)}$. Along with each value v for every sub-puzzle the solution that is sent back to the server consists of t and the message m that was sent to the server when asking for a puzzle. I think you may have misunderstood the client puzzle protocol that the paper you linked introduces, as it is quite different than yours. I'll use the example given in section 4.1.1:

Code generValue: 426305d23afb6789d840e47ffdbbf300000000

yFull: bb85bcab87647fcbe48e8bebd8elcgccacd9569a132cca948b7aef3968adbc29

puzzle: y: bcab07647fcbe40e8bebd8e1c9ccacd9569a132cca948b7aef3968adbc29

z: a9b2Hb134a481fra3dFfb55celeeld6917807beb3cbd52de80d9Facld348fce

In this example, generValue is the server secret 426305d23afb6789d840e47ffdbbf3 appended with the counter-nonce 00000000, which is used to produce puzzles. yFull is the hash of generValue (serverSecret+nonce). z is the hash of yFull. y is yFull with the first 2 bytes (16 bits) removed. The

amount of bits removed from y_{Full} is the difficulty level. The client's goal is to find what bits were removed, by hashing every 16 bit sequence appended to y . The client knows that it has found the 16 bits after its hash matches z .

This is different from your client puzzle, where the goal is simply to find a value whose hash is a particular pattern of 24 bits.

I'll ask you this: is every possible pattern of 2^{24} bits guaranteed to show up in the hash of any of 2^{24} inputs? If not, then what makes a pattern of 24 zeros or 24 ones special than any other 24 bit pattern (which may not show up in the hash)?