Anti Brute Force Resource Metering

Helping to Restrict Web-based Application Brute Force Guessing Attacks through Resource Metering

Abstract

Web-based applications authentication processes are frequently vulnerable to automated brute force guessing attacks. Whilst commonly proposed solutions make use of escalating time delays and minimum lockout threshold strategies, these tend to prove ineffectual in real attacks and may actually promote additional attack vectors.

Resource metering through client-side computationally intensive "electronic payments" can provide an alternative strategy in defending against brute force guessing attacks. This whitepaper discusses how such a solution works and the security advantages it can bring.

Author

Gunter Ollmann, Professional Services Director, NGS – email: gunter [at] ngssoftware.com

Section 1: **Background**

For most web-based applications that require customers to uniquely identify themselves prior to granting access to key functional aspects of the online system, a solid and reliable authentication process is the primary security barrier. When these applications are providing online services to a large and/or diverse customer base, the authentication process must be able to withstand an increasing number of malicious attack vectors. Poorly designed or implemented authentication processes are easily exposed and as a consequence are likely to result in subsequent exploitation resulting in an increase in adverse public scrutiny and a concomitant decrease in customer confidence.

A critical element to any successful online authentication solution is the way in which account lock-out processes are implemented. In particular, how the application handles an attacker attempting to guess the login credentials of customers and prevents access during automated brute-force guessing attacks. For an attacker, authentication solutions that unintelligently lock-out access to customer accounts after a specific threshold has been reached (typically three attempts) can be easily turned into a highly successful denial of service attack. By quickly reaching the threshold limit of bad password attempts, the attacker can cycle through customer names or account numbers and lock-out each in turn.

The speed at which these attacks can be conducted can be remarkable. It is not uncommon for attackers to conduct as many as 200 login attempts per second over standard DSL connections. In some cases, when applications have very simple single-tier authentication processes and the attacker has good network access to the host, they can achieve many simultaneous parallel connections – rates of between 1500-10,000 login attempts per second can be achieved.

While various solutions for handling authentication processes and account lock-out policy have been suggested in an attempt to manage these attack vectors, it is left to each individual organisation to carefully weigh up the pro’s and con’s of each option before arriving at a compromise solution.

One frequently considered compromise is to link authentication failures to a specific IP address and to conduct multiple actions based upon this information. Unfortunately, due to the use of ISP proxies and network address translation, it is highly likely that legitimate customers will seek to access an application from behind these devices and that therefore their connections will appear to come from the same IP address. Subsequently, any automated response to block or otherwise inhibit IP addresses associated with an attack may consequently prevent legitimate customers from accessing the application.

However, the use of IP address information as part of an automated defence strategy can still be valuable. Instead of focusing upon the outright blocking of an attackers IP address, organisations may wish to adopt a strategy capable of slowing down an automated attack to such a degree that it becomes an unviable attack vector and therefore forces an attacker to direct their attention to alternative “softer” targets – preferably some other organisation. By requiring an “electronic payment” for each login attempt, the application can enforce a computational overhead (and subsequent time delay) upon the submitting host which can then be used as a form of resource metering. A similar strategy has proved successful in reducing email-based Spam attacks – where the “electronic payment” overhead is referred to as “hashcash”.

Section 2: **Resource Metering**

Resource metering is a technique designed to restrict the repetition frequency of data submission to an application or host system. The ability to reduce the frequency of data submissions can play an important role in controlling an attacker's ability to conduct an automated brute force guessing attack. To be successful, a resource metering solution should enforce restrictions at the client-side and not consume additional resources at the server-side.

The most practical method of implementing resource metering is through the use of cryptographic hashes. The use of a cryptographic hash in this fashion is sometimes referred to as requiring an "electronic payment" before processing the customer's submission. In essence, the server-side application requires the customer's client to compute a value that is computationally intensive, but easy to validate, before processing the submitted data.

The principles are based upon the fact that there are numerous mathematical problems that are easier to verify than they are to compute. For example, the calculation of square roots of a large number is a complex and processor intensive task – but is easy to validate since:

$$y = \sqrt{y} \times \sqrt{y}$$

While the concept of using square root computations sounds appealing, unfortunately computers are very fast and therefore to be conducive to resource metering the number would have to be 1000's of digits long. Given the HTTP medium, it would probably take longer for the transmission of the data than the time taken to compute the values.

2.1. Learning from Spam

Email has been around longer than the web – consequently there are a number of security solutions that have had a longer gestation period in which to mature and prove their worth. The implementation of resource metering in the fight against email-based Spam has already proven to positively reduce the success of these automated intrusions – normally resulting in Spammers seeking softer or alternative targets.

The most important resource metering solution within the SMTP domain is commonly referred to as "hashcash". By requiring the attacker's email agent to conduct a computationally intensive process as part of each email submission, the spammer incurs a delay with each email delivery and greatly reduces the effectiveness of their intrusion. Whilst not completely eliminating the threat, "hashcash" helps to restrict the flow of Spam and can force the malicious spammer to seek a more vulnerable system.

Borrowing heavily from the concept of "hashcash" within email services, it is a relatively simple process to include a similar level of resource metering within a web-based applications authentication process – the purpose being to greatly reduce the effectiveness of an automated brute force guessing attack.

2.2. Hashcash

Originally proposed by Cynthia Dwork and Moni Naor in 1992, and independently invented in May 1997 by Adam Back, the concept of hashcash was introduced as a way of installing a resource metering solution (referred to as an "electronic payment") capable of throttling the systematic abuse of services such as email and anonymous re-mailers.

As mentioned previously, there are a number of mathematical problems that are far easier to verify than they are to initially compute. "Hashcash" makes use of a mathematical principle called "partial hash-collisions". These partial hash-collisions require smaller variables than those required for the equivalent resource metering by square root calculations (making them more convenient for inclusion in server responses), are substantially faster to verify, and often simpler to program.

"Hashcash" traditionally computes hash-collisions based upon the email recipients address, date and some random seed data.

2.2.1. Hash Functions

A hash function is a one-way cryptographic function that transforms a string of characters, usually into a shorter fixed-length value or key, based upon the assumption that it is extremely difficult and time consuming to find two strings capable of producing the same output. Given a change in the input string, the resultant hash also changes. Common hash functions include MD5 and SHA-1. "Hashcash", as implemented in numerous anti-spam email solutions, makes use the SHA-1 hash function.

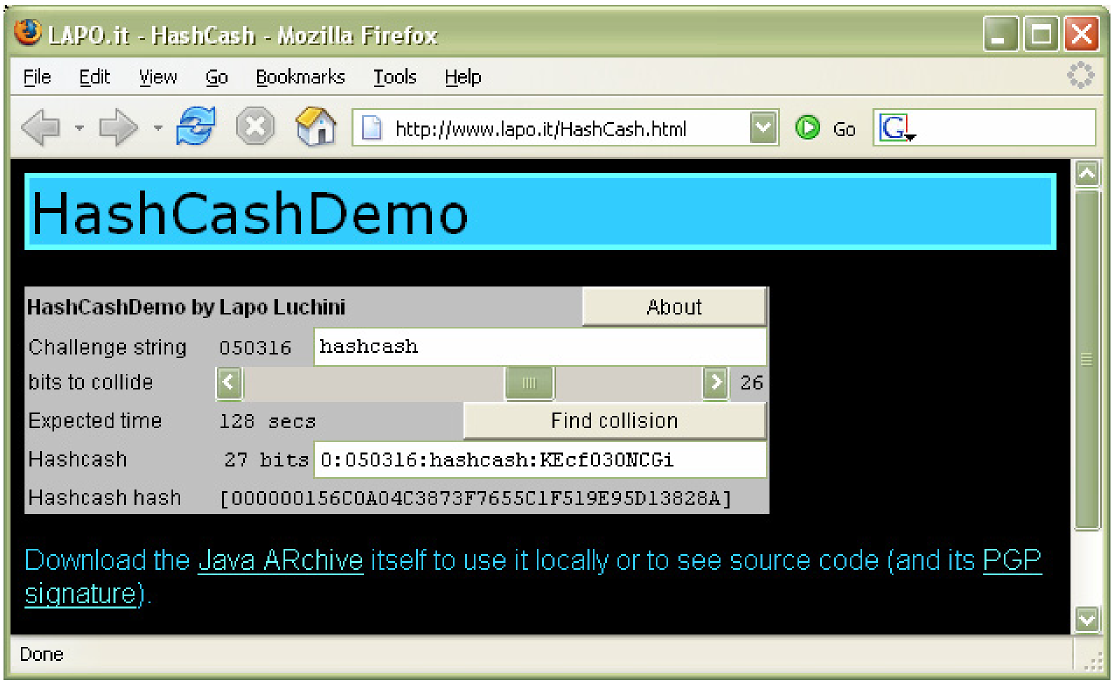
Hash functions such as SHA-1 have been designed to be collision resistant – i.e. two distinct input strings should not produce that same hash value. In fact, the SHA-1 cryptographic function is not expected to result in the same hash result for two distinct input strings for 2^{160} different values.

2.2.2. Partial Hash-collisions

In theory, calculating a full "hash-collision" is computationally infeasible – requiring great computing resources and extended timescales (although shortcut vectors for finding collisions have been announced in the common hashing functions: SHA-0, SHA-1, MD4, MD5, HAVAL-128, and RIPEMD).

"Hashcash" makes use of partial hash-collisions (sometimes referred to as *n*-bit collisions). Instead of requiring the hashed value from one string to be completely the same as the hashed value of a different string (or indeed calculating the same string), only parts of the two hash values need to be identical. By defining how much of the first hash value must match the other, it is possible to control how much computational time must be expended in evaluating a correct answer.

For example, using an online "hashcash" demo supplied by Lapo Luchini and implemented in Java, calculation of the first 21 most significant bits takes around 4 seconds on a 3GHz processor desktop computer – while calculating the first 27 most significant bits takes a little over 2 minutes.


LAPO.it - HashCash - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.lapo.it/HashCash.html
Go
About
Find collision
Done

HashCashDemo

HashCashDemo by Lapo Luchini

Challenge string	050316	hashcash
bits to collide		26
Expected time	128 secs	
Hashcash	27 bits	0:050316:hashcash:KEcf030NCGi
Hashcash hash	[000000156C0A04C3873F7655C1F519E95D13828A]	

Download the Java ARchive itself to use it locally or to see source code (and its PGP signature).

Figure 1: Screenshot of a Java-based "hashcash" calculation at http://www.lapo.it/hashcash.html

Section 3: **Web-Authentication Resource Metering**

The purpose of introducing an "electronic payment" to the authentication process is to provide a degree of time control over which customers may submit their login credentials to the application server – therefore providing a mechanism for resource metering. Additionally, by forcing a computational overhead to occur at the client-side host, the application server is free to process other application processes.

3.1. Incremental Timeouts

A well-discussed and increasingly popular strategy for handling automated brute force guessing attacks against an application's authentication process is through the use of incremental timeouts. Essentially, with each failed login attempt, the application server takes longer to respond to the attacker (or customer).

Consider an authentication process that requires the customer to fill in their user ID and password on an HTML-based form and submit this data to the application for authentication. If the login credentials are incorrect, the server redirects the customer to the original login page after a couple of seconds telling them that they have failed and must retry. If the customer supplies incorrect login details a second time, the response from the server takes 4 seconds longer – and so on until some maximum threshold is reached (typically no more than 2 minutes). This incremental timeout strategy may be combined with a maximum login attempt threshold – at which, after say 6 incorrect login attempts, the customer account is locked out and no longer available.

There are a number of problems with this incremental timeout strategy:

- The timeouts are typically linked to SessionID's supplied to the client browser when the customer first connects to the application. An attacker can instead reject SessionID's if they become associated with an unacceptable timeout and make a new connection – therefore acquiring a new untainted SessionID without the overhead of a timeout.
- Alternatively, application designers may choose instead to link the incremental timeouts to each unique customer's user ID. Instead of trying to brute force guess the password associated with a specific user ID, the attacker decides upon a fixed password and cycles through user ID's to discover customers that have chosen to use that particular password. In this attack, the attacker is never subjected to any incremental timeouts.
- If the application developer chooses to base their incremental timeout strategy on the connection IP address – it is likely that any legitimate customers attempting to access their own accounts from behind a proxy (AOL and many dial-up ISP's use these) or NAT firewall (many corporate environments implement this) shared with the attacker will be subjected to similarly long timeouts. In addition, it is not uncommon for the proxy servers of large ISP's to be load balanced – meaning that there is no guarantee that the attacker will consistently access the application from the same IP address.
- In a load-balanced environment it is difficult to enforce server response timeouts as the attacker may simply break their connection with one server and establish a new connection with a different server.
- With increasing frequency, attackers are making use of Botnets (essentially numerous home PC's and workstations that have previously been compromised and had Trojan horse software installed on them) to conduct large-scale brute force attacks from multiple sources. Therefore, the attack may be spit up and distributed amongst the numerous hosts of the botnet and not just come from a single IP address source.

However, by using a carefully designed and implemented "electronic payment" resource metering solution, it is possible to utilise a resource metering strategy without being subject to many of the limitations listed above.

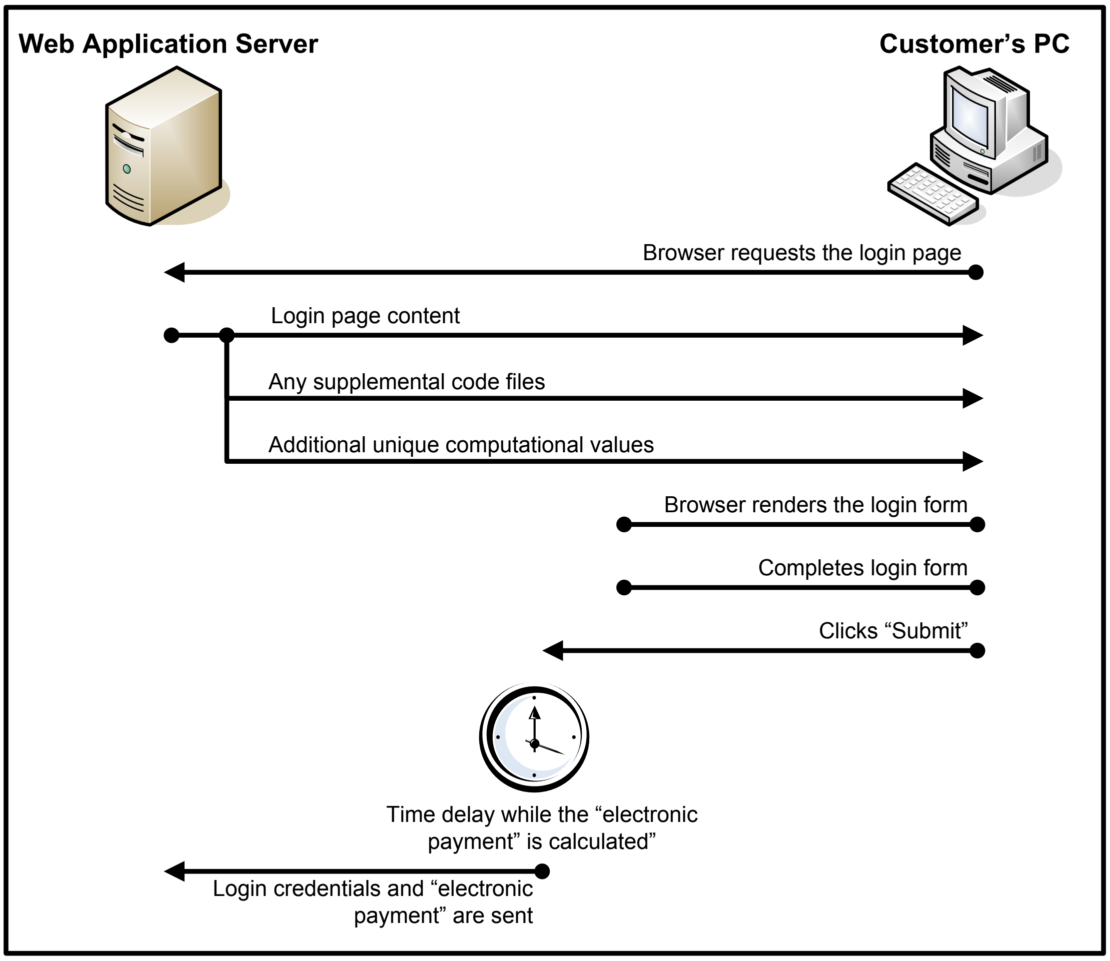
3.2. Understanding Web-Authentication Resource Metering

Typically, in order to implement a resource metering solution, it is necessary for the client browser to be capable of performing mathematical functions via a scripting language. Therefore, resource metering cannot be used in the authentication process for web-based applications designed specifically not to require client-side scripting.

In the majority of cases, depending upon the mathematical routine(s) used to create the "electronic payment", any popular client-side language may be used (e.g. Java, JavaScript and VBScript).

The sequence of events in this process is:

1. The client browser requests the application login page,
2. The server sends the HTML login page which also contains the JavaScript mathematical routine necessary for calculating the "electronic payment" and any necessary computational values. Alternatively the mathematical routine could be held in a separate downloadable JavaScript file (*.js).
3. The client browser then renders the HTML page, presents the login page to the customer, and prompts them to supply their authentication credentials.
4. Once completed, the customer clicks the "submit" button – at which stage the JavaScript computational routines are executed and the "electronic payment" is calculated.
5. The client browser then submits this calculated value along with their authentication credentials.
6. The server first validates the calculated value and, if the value is correct, proceeds to process the submitted login credentials.


Web Application Server
Customer's PC
Browser requests the login page
Login page content
Any supplemental code files
Additional unique computational values
Browser renders the login form
Completes login form
Clicks "Submit"
Time delay while the "electronic payment" is calculated"
Login credentials and "electronic payment" are sent

If the "electronic payment" value is missing or incorrect, a number of options exist for handling the customer and probable attacker:

- Redirect the customer back to the login page and start the entire process again using the same mathematical routine and different computational values,
- With each failed submission the computational values become longer or more time consuming to calculate,
- To prevent any external modification or optimisation of the mathematical routine it could be selected at random from a library of pre-created routines (only one routine should be sent within the HTML page at any one time) or rotated between multiple calculating engines,
- The mathematical routines could become more sophisticated and require a higher "electronic payment" for each failed or repeated login attempt.

3.3. Design Decisions with Resource Metering

A number of options exist when selecting and building a resource metering solution designed specifically to protect the web-based authentication process. Dependant on the nature of the application and the likely customer audience, a number of design considerations should be made:

- Mathematical algorithm selection – The choice of mathematical algorithm(s) used to create the "electronic payment" is critical. It must operate in a way that granular control of "payment" thresholds is possible and can therefore cover the minimum and maximum resource metering thresholds necessary to provide protection – typically 0 to 300 seconds.

- Simple and mathematically robust – the chosen client-side algorithm must be mathematically robust so that computational shortcuts cannot be achieved, or are unlikely to greatly affect the speed in calculating the "electronic payment". At the server-side, the chosen algorithm must be capable of receiving a customer "payment" that can be easily and rapidly validated. Consequently it is recommended that a well known cryptographic algorithm be used.

- Minimum "electronic payment" thresholds – developers should decide upon an initial or minimum "payment" threshold. In many cases, an initial "payment" calculation delay of 1 to 3 seconds is to be recommended.

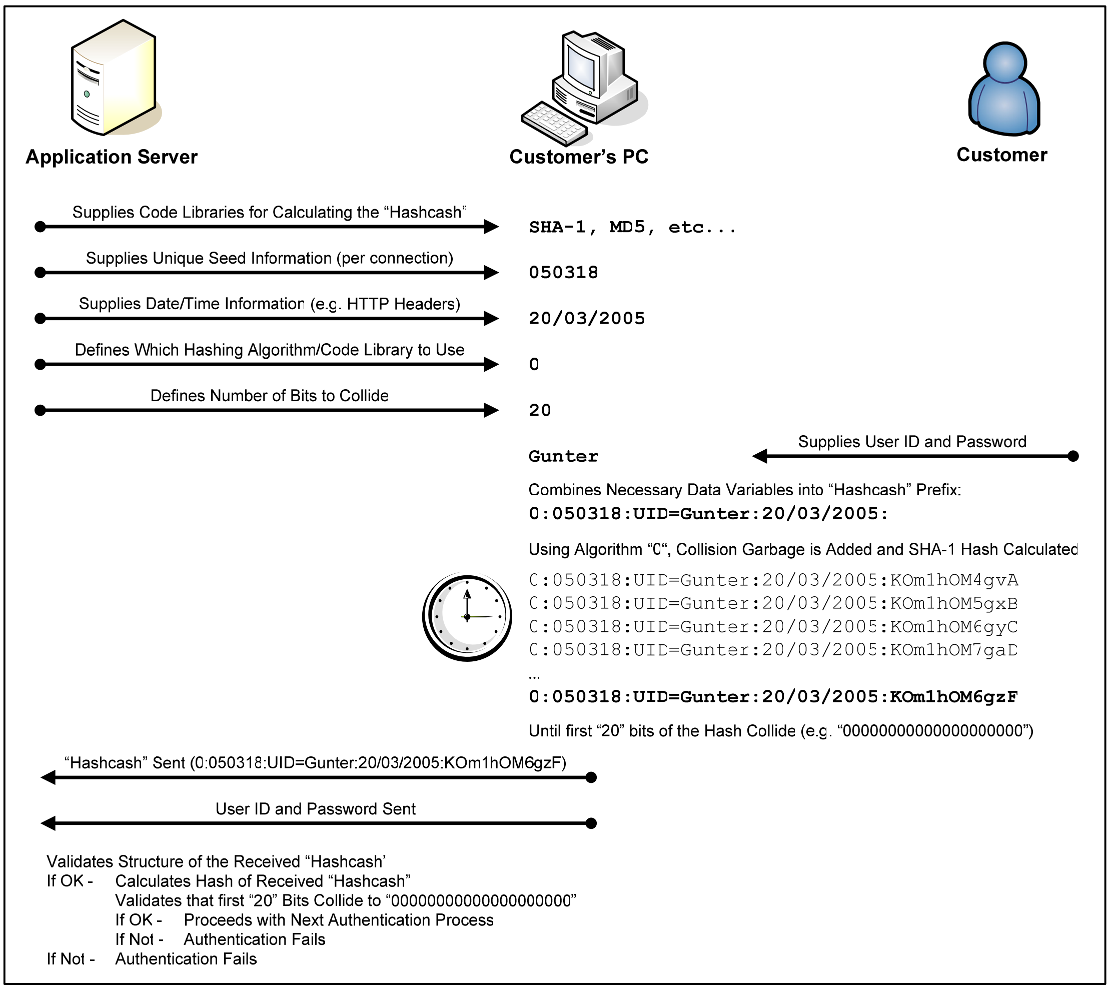
- Maximum "electronic payment" thresholds – dependant on whether any other anti brute forcing mechanisms are included within the authentication processes; a maximum "payment" calculation delay of 2 to 3 minutes is normally recommended.

- Download size – the combined file size of the HTML code content along with any necessary file inclusions and algorithms should be considered. While focused attacks are common, attackers are the minority and the time taken for the client browser to download all page content should not become prohibitive over slow Internet connections.

- Fit with account lock-outs – automated account lock-out processes are likely to influence the way a resource metering solution is to work. Developers should review the number of incorrect login attempts necessary to trigger an account lock-out event, and then decide upon the appropriate intermediary thresholds for each "electronic payment". In general, increasing the threshold by 50% or a doubling of the time to calculate the "electronic payment" is to be recommended – until the maximum threshold is reached

3.3.1. Hashcash in Web Authentication

Extending the lessons learned in implementing anti-spam "hashcash" solutions, it is a trivial task to adopt similar formatting and cryptographic algorithms and use them as the key resource metering process for web-based application authentication systems. The following example shows one way in which "hashcash" could be implemented.



Supplies Code Libraries for Calculating the "Hashcash" → SHA-1, MD5, etc...

Supplies Unique Seed Information (per connection) → 050318

Supplies Date/Time Information (e.g. HTTP Headers) → 20/03/2005

Defines Which Hashing Algorithm/Code Library to Use → 0

Defines Number of Bits to Collide → 20

Gunter ← Supplies User ID and Password

Combines Necessary Data Variables into "Hashcash" Prefix:

```
0:050318:UID=Gunter:20/03/2005:
```

Using Algorithm "0", Collision Garbage is Added and SHA-1 Hash Calculated

```
0:050318:UID=Gunter:20/03/2005:KOm1hOM4gvA
0:050318:UID=Gunter:20/03/2005:KOm1hOM5gxB
0:050318:UID=Gunter:20/03/2005:KOm1hOM6gyC
0:050318:UID=Gunter:20/03/2005:KOm1hOM7gaD
...
0:050318:UID=Gunter:20/03/2005:KOm1hOM6gzF
```

Until first "20" bits of the Hash Collide (e.g. "00000000000000000000")

← "Hashcash" Sent (0:050318:UID=Gunter:20/03/2005:KOm1hOM6gzF)

← User ID and Password Sent

```
Validates Structure of the Received "Hashcash"
If OK -   Calculates Hash of Received "Hashcash"
          Validates that first "20" Bits Collide to "00000000000000000000"
          If OK -   Proceeds with Next Authentication Process
          If Not -  Authentication Fails
If Not -  Authentication Fails
```

It is important to note the following about this sample "hashcash" implementation:

- A separate code library (e.g. a Java class file) is downloaded by the customers PC. This file may contain multiple cryptographic routines capable of calculating several different types of "hashcash" electronic payment. The specific routine to be used in the authentication process will be selected by the application server, and may be varied between login attempts to help prevent pre-computation issues or shortcut optimisations.
- For each and every customer connection, a different "seed" value is supplied. This seed value is used as part of the "hashcash" calculation process. It is important that this seed value be sufficiently long enough to ensure that an attacker could not viably pre-compute the hashes in advance (i.e. the "050318" in the example above could be longer and composed of alphanumeric values).
- The application must keep a record of which seed value is assigned to each connection. In many cases this connection monitoring can be maintained through standard SessionID handling processes.
- The inclusion of date or time information is included to make it more difficult for an attacker to pre-compute all hashes in advance. This can also be complicated by the fact that the server may also require a different hashing algorithm or computational code segment be used to calculate the "Hashcash". For added complexity, the use of hour and minute data within the "hashcash" prefix would be recommended.
- The server decides upon the number of bits necessary for a partial "hash-collision". This value is the key to defining how much computational effort (and consequently time delay) must be incurred by the customer's client in calculating the electronic payment.

- The customer's client must append collision garbage (random data) to the prefix and calculate SHA-1 hashes until it comes across a "hashcash" string that result's in a partial "hash-collision". In this example, a partial hash-collision occurs when the first 20 characters of the hash are all zero – i.e. "00000000000000000000". Developers could of course define any value they wish to collide with. Using the SHA-1 hashing algorithm coded in Java, a 20 bit collision took about 2 seconds.
- After the "hashcash" value has been calculated, it is submitted to the application server along with the other authentication data. The application should validate the structure of the "hashcash" first before calculating the corresponding hash. If the appropriate partial hash-collision occurs, the application can then proceed with processing the authentication information (e.g. User ID and password).
- Should the customer fail to authenticate, the application would require the user to attempt to login again. However, in this and subsequent failed attempts, the server would assign a new "seed" value and also increase the number of bits necessary to cause a partial hash-collision (i.e. increasing the time it takes to calculate a valid "Hashcash" value).

3.3.2. Resource Metering Recommendations

Should an organisation choose to implement a resource metering solution, the following recommendations are made:

- Well known mathematical cryptographic hashing algorithms are recommended as they are likely to require less dynamic data to be transported between client and server for the process of calculating the "electronic payment". Additionally, since these algorithms have received a lot of independent analysis, they are likely to be more robust against computational shortcuts.
- The calculation of the "electronic payment", when using a partial "hash-collision" algorithm, should include a random seed value supplied by the application server (which forms a unique identifier for the connection, and is changed with each connection or refresh of the login page).
- Developers should carefully review and test the time necessary for a customer's client to calculate valid partial hash-collisions for each value likely *n*-bit collision value. If multiple calculation routines are used, each routine should be tested. Developers should be aware that older computers may be slower in calculating hashes.
- Given that there is a delay between submitting the login credentials, calculation of the "electronic payment" and final response from the server; the customer should be provided with a visual effect to indicate that things are proceeding as planned and that they do not need to re-submit the data or refresh their browser page. The use of progress bars and animated timers are recommended.

3.4. Resource Metering and Brute Force Attack Vectors

Resource metering can provide a valuable method of preventing or otherwise limiting the susceptibility of a web application's authentication process to multiple brute force guessing attack vectors. Unlike implementations that rely on incremental timeouts being enforced at the server-side or enforcing access threshold limitations, the use of client-side computationally intensive "electronic payments" can provide a more elegant and robust solution. Key points to a resource metering solution include:

- Load Balancing - Since the resource metering effectively occurs at the client side, it is not necessary for the application to manage submission timeouts. Therefore, with load balanced applications, it is not possible for an attacker to bypass server response delays by requesting data from alternative load balanced hosts.
- Multi-threaded Attacks – in most web-based brute force guessing attacks, the attacker makes multiple simultaneous connections to the application server; thus conducting a multi-threaded attack. In doing so, the attacker can conduct hundreds or thousands of guessing attempts every second. By forcing an attacker to calculate

and submit an "electronic payment" with each guess, even using an algorithm with partial hash-collisions that result in only 1 second of delay, the attacker may only carry out one guess per second irrespective of how may simultaneous connections they have from the same attack client (since the CPU often reaches 100% utilisation when calculating the "hashcash", calculating 200 simultaneous values means that each thread has a 1/200th share of the CPU's computational ability, and correspondingly it will take 200 times longer to calculate a standard "1 second" value).

- Incremental Control – Since time delays and computational effort are controlled by a collision bit length, it is a simple process to increase the time spent in calculating the "electronic payment" without any subsequent increases in server-side session management requirements.
- Unbounded payment - These incremental steps in "electronic payment" are not bound to a single customer's login variables. Consequently, the same payment is required whether an attacker is brute force guessing the customers User ID, Password, or any other authentication value.
- IP Address Restrictions – By enforcing "electronic payments" on all authentication submissions (even without the use of incremental controls), the application could also enforce different levels of payments for different IP ranges. For instance, IP addresses or ranges associated with DSL users in China could require "hashcash" that takes 10 seconds to calculate, while well known ISP Proxies could incur 5 seconds and trusted IP addresses only 1 second of payment.
- Script Requirements - This solution requires that the customer's client browser supports an appropriate scripting language.
- Computer Specifications - The specification of the customer's client host will dictate the speed at which it can calculate mathematical challenges and consequently the "electronic payment" may be less for new or high end computers.
- Alternative Applications – Resource metering through "electronic payments" does not have to be limited to authentication processes. Organisations may find that implementing a similar strategy for financial transactions or any other form of data submission back to the application server could benefit from its use and the corresponding throttling of speed.

3.4.1. Why not use Mandatory Server-side Timeouts?

Some organisations may have chosen to adopt a mandatory server-side timeout strategy to implement a form of resource metering against brute force attacks. In these schemes, the application server forces all authentication submissions to wait some period of time before responding (e.g. the customer clicks "submit" – the data is sent to the server – the server processes the data and waits 4 seconds before issuing any response to the customer). However, there are a number of limitations to this strategy which may be overcome through the use of client-side imposed resource metering.

Most importantly, by forcing computationally intense processes to occur at the client-side (not just a wait or sleep-type function) it becomes almost impossible to conduct a multi-threaded brute force attack. Consider an application that enforces a 4 second delay at the server-side for each submission. If an attacker opens 1000 simultaneous connections to the server and submits a different customer authentication to each one, after 4 seconds 1000 responses are received by the attacker – corresponding to 250 guessing attempts per second. Using the earlier "hashcash" example but with a 4-second computational load, if the attacker makes 1000 simultaneous connections to the server, it will take approximately 4000 seconds to submit all the guesses – corresponding to 0.25 attempts per second.

Finally, by shifting the onus away from the server environment and on to the client, valuable resources are freed up (including processing overheads, threads, memory allocations and open TCP ports).

Section 4: **Conclusions**

The enforcement of resource metering through the use of “electronic payments” is likely to provide valuable protection against many forms of brute force guessing attack vectors commonly employed against web-based authentication systems.

Implementing a system capable of enforcing “electronic payments” is a relatively simple process and likely to result in an immediate strengthening of an application’s security robustness. Implementation lessons learned from anti-spam solutions such as “hashcash” provide a valuable insight in selecting and tuning a resource metering solution. Strategies which incorporate partial “hash-collision” calculations provide an effective granular control over the necessary “electronic payments”, without incurring greater server-side and application functionality overheads.

As an anti brute force guessing protection system, resource metering provides several advantages over more traditional strategies such as fixed or incremental time-outs and account lock-out thresholds. In almost all cases, the system would appear transparent to the customer (and certainly cause no more delays than a traditional submission time delay solution) and is capable of severely limiting automated attacks.

Resource metering through “electronic payments” is recommended for web-based applications that have large customer bases, and/or are likely to be targeted for attack with automated brute force guessing tools. Similarly, organisations that have encountered problems through account lock-out denial of service attacks are also encouraged to review the possible benefits of implementing a resource metering solution.

4.1. References

Web - http://www.hashcash.org/papers/hashcash.pdf

Web - http://www.lapo.it/HashCash.html

Cynthia Dwork and Moni Naor. “Pricing via processing or combating junk mail” In Proceedings of Crypto, 1992.