Preventing session hijacking in collaborative applications with hybrid cache-supported one-way hash chains

Amerah Alabrah
Department of Electrical Engineering and Computer Science
University of Central Florida, Orlando, Florida
College of Computer and Information Sciences
King Saud University, Riyadh, Saudi Arabia
amerah@knights.ucf.edu

Mostafa Bassiouni
Department of Electrical Engineering and Computer Science
University of Central Florida, Orlando, Florida
USA
bassi@cs.ucf.edu

Abstract— Session hijacking attacks of social network websites are one of the commonly experienced cyber threats in today's Internet especially with the unprecedented proliferation of wireless networks and mobile applications. To address this problem, we propose a cache supported hybrid two-dimensional one-way hash construction to handle social networks' user sessions authentication in collaborative applications efficiently. The solution, which presents a major redesign from [18], is based on utilizing two-dimensional OHC chains equipped with sparse caching capabilities to carry out authentication during social networks users' sessions. We analyze the proposed hybrid scheme mathematically to determine the cost of authentication and develop a quartic equation to check the optimal configuration of the two dimensions. We also evaluate the hybrid scheme with simulation experiments of different configurations and scenarios. The results of the simulation experiments show that the hybrid scheme improves performance of the OHC tremendously while efficiently and securely handling authentication.

Key words: *Internet Sessions, Collaborative environments, Wireless networks, Mobile devices, one-way hash, authentication*

1 Introduction

Session hijacking attacks are considered one of the commonly experienced cyber threats in today's Internet. These attacks not only impact users and service providers alike, but can certainly jeopardize the whole Internet experience. In a 2010 Open Web Application Security Project (OWASP) [1], broken authentication and session management attacks were identified among the top ten list along with injection, cross-site scripting and cross-site forgery attacks. In their 2013 Release, OWASP still lists these attacks, which lead to session hijacking, among the top 10 security threats of the Internet [2]. Additionally, the broken authentication and session management attacks have moved up in prevalence to be second in the list.

Session hijacking typically targets the Hypertext Transfer Protocol (HTTP) where session based communication is used to keep a user/browser state. Before a session is established, many websites and especially collaborative applications use the secure HTTPS connection to grant users access to their services. During the HTTPS connection, user login is established and login credentials are replaced by session identifiers such as session cookies as a cheaper alternative to the secure HTTPS particularly in mobile devices characterized by their limited computational abilities. Such threat is more prominent in collaborative applications networks, particularly with the *remember me* option, which extends the users' sessions to unpredicted periods. The rise of session hijacking attacks in collaborative applications and other Internet applications is attributed to the increased utilization of HTTP cookies in session authentication in lieu of the session-wide employment of the secure HTTPS connection. Session cookies are typically stored on the clients' machine and are usually transmitted over unsecure wireless connections, thereby compromising the client's social network experience if the cookies are illegally accessed whether through passive attacks (e.g. eavesdropping) or active attacks (e.g. cookie stealing). Many tools are available to carry out session hijacking attacks which include CookieCatcher [3], FaceNiff [4], Firesheep [5] and many more. Session hijacking can be either active where attackers take over the whole session and impersonate the social network's victim, or passive which involves sniffing out a session and passively watching traffic.

In this paper, we present a hybrid scheme that utilizes one-way hashing and sparse caching techniques. Our objective is to propose a scheme that can be easily deployed in an efficient and secure manner that does not burden collaborative applications providers or incur extraneous memory resources on users' platforms.

The remainder of the paper is organized as follows. Section 2 provides a brief overview on previous literature on this area. In section 3, we introduce the fundamental concepts for this paper. In section 4, we provide a detailed description of the proposed scheme highlighting its main features. In section 5, we introduce comparisons and tradeoffs and evaluate the performance of the proposed scheme. Finally, section 6 concludes the paper.

2 Related Work

Recent research on attacks targeting collaborative applications and social media networks has focused on detection schemes (e.g. [6]), reputation attacks (e.g. [7], [8]); both of which are directly related to session hijacking

978-1-63190-043-3 © 2014 ICST
DOI 10.4108/icst.collaboratecom.2014.257327

attacks. The detection schemes do not, however, provide prevention against such attacks, and so do the reputation attacks solutions. Research dealing with session hijacking threats' prevention is vast and the issue has received attention from a variety of perspectives. The authors of [9] indicate that the current IEEE 802.11 wireless networks are vulnerable to session hijacking attacks as the existing standards fail to address the lack of authentication of management frames and network card addresses, and rely on loosely coupled state machines. Several proposed schemes have tried to solve this issue. Liu et al. [10] propose the use of a secure cookie that relies on *HMAC* to ensure the authenticity of transmitted cookies. However, due to its reliance on the Secure Socket Protocol (SSL), this solution does not provide scalability. Solutions not utilizing HTTPS such as SessionLock in [11], SessionShield [12] both have shortcomings. SessionLock uses session fragment identifier and *HMAC* to create authentication tokens exchanged between the server and the client throughout the life of the session. On the other hand, SessionLock does not provide strong protection against active session attacks since it aims at providing protection against passive attacks like eavesdropping. SessionShield [12] employs a proxy like application at the client's side to prevent session hijacking. Nonetheless, this technique presents problems pertaining to scalability and compatibility.

To secure communication in an internet session, cryptographic techniques such as one-way hash chain (OHC) technique that rely on one-time passwords proposed by Lamport [13] have been utilized. In particular, the OHC technique has been employed in many application with the aim of mitigating the potential of session hijacking. For example, the authors in [14] proposed using One-Time-Cookies (OTC), where disposable credentials called one-time cookies replace authentication credentials. The OTC scheme generates a set of tokens that are only used once and discarded once used. To overcome some inherent drawbacks in the OHC scheme such as high computational overhead, the authors of [15] and [16] propose devoting some memory in the client's machine for previously computed hashed values for authentication tokens that are fetched as needed. This relieves some of the computational overhead, but requires additional memory that sometimes is not available especially with low-end devices such as some wireless devices. Another approach suggested in [17] and [18] is to divide the one-way hash chains in a two-dimensional format. Compared to the sparse caching approach, dividing the OHC into multiple smaller chains does not provide a similar reduction in the computational overhead. In this paper, we propose a solution that benefits from the advantages of the sparse caching strategy proposed in [15] and [16], in addition to the efficiency of the two-dimensional one-way hash construction [17] and [18]. The hybrid approach is light, efficient and easy to implement as a client side plug-in.

3 PRELIMINARIES

The basic assumption of the OHC scheme is that an authentication token derived by applying a one-way hash function is used to protect session cookies. Essentially, we use a one-way hash chain of length *N*, which corresponds to the number of transactions in a session, to protect a session from being sniffed or hijacked. Initially and using an HTTPS channel, some parameters are exchanged between the server and the client. These parameters include a shared secret S_0 in addition to the length of the Internet session denoted *N.* Based on such parameters and using the agreed upon cryptographic hash function, the OHC protects the kth transaction with an authentication token $U_k=H^{N-k+1}(S_0)$ derived by applying the cryoptographic hash function m times. Thus, for the notation $H^2(x)= H(H(x)),$ the hash function is applied twice on x. To take an example, if N=100, the authentication tokens for the 1st, 2nd, and 3rd transactions are $U_1=H^{100}(S_0)$, $U_2=H^{99}(S_0)$, $U_3=H^{98}(S_0)$, respectively.

Inherently, the OHC approach suffers from a high computational overhead caused by the need to compute the hash function recursively in the first iteration. Thus, the number of transactions expected to be handled during an Internet session is a key factor contributing to the computational overhead required to compute the hash function. Basically, without accurate statistics of users' behavior, which tends to vary tremendously, the number of transactions can be overestimated, thereby unnecessarily increasing overhead, or underestimated resulting in having the user redo the login process. The solutions proposed in [15] and [16] address this problem by devising sparse caching units in which the hashed secrets are pre-computed, stored and fetched as needed. However, while the solution gives acceptable performance in reducing the computational overhead, cache memory is scarce in many mobile devices and the space for cache in these devices cannot be committed for a long time. Alternately, the authors of [17] and [18] propose a scheme that deploys two-dimensional mini one-way hash chains to significantly reduce the overhead of OHC without deploying cache memory.

We propose a hybrid solution that maximizes efficiency and minimizes the cost of memory resources. To achieve this, we divide the one-way hash chain into multiple chains and support them with caching units where authentication tokens are stored and fetched as needed. To measure efficiency, we use the number of hash operations needed in a session. In the next section, we define the configuration of the system's components and determine the cost based on these configurations. Before this overview is presented, we introduce the notations used in the scheme. We refer to the proposed scheme as the hybrid scheme.

Scheme Notation
I = mini OHC scheme
J = OHC Caching scheme
K = Hybrid scheme

Common Notation
N = number of transactions
X = horizontal chain for seeds
Y = vertical chain for authentication tokens
M = space interval between cache units
$N = X \times Y$ // simplified assumption

We will introduce a more detailed description of these notations in our description of the schemes below.

A. The mini OHC Scheme:

The conventional OHC scheme has one dimension where one seed is used to generate authentication tokens by a single one-way hash chain for the whole session. However, the mini OHC scheme is arranged into two dimensions (see Figure 1). In the first dimension (i.e. the horizontal axis X_i), there is a single hash chain that computes the seeds for the second dimensions chains (i.e. vertical axis Y_i). In the second dimension, we have multiple hash chains that use these seeds to generate authentication tokens. Authentication tokens are generated by hashing the seeds using cryptographic hash functions (e.g. SHA-1, SHA-2 or SHA-3). These cryptographic hash functions are known for their resistance against attacks.

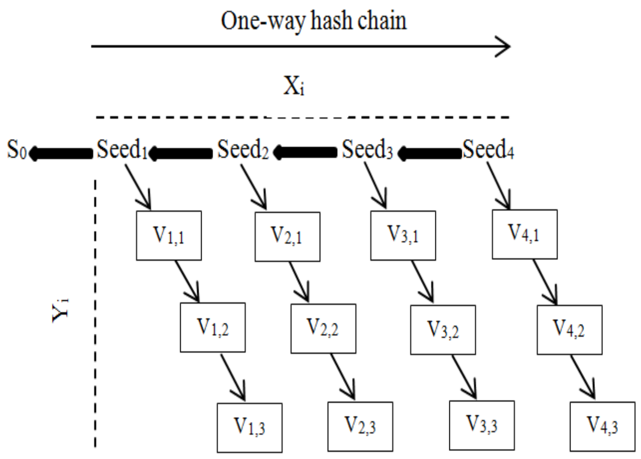

Figure 1 mini OHC construction

Authentication in the mini OHC is done in three steps:

Initialization:

The server and the client utilize an HTTPS channel to exchange the number of transactions in a session N, an initial value of the shared secret S_0, and the length of the authentication token chain Y_i. Based on these variables, the number of seeds is determined, and $Seed_1$ is calculated to be used in the first authentication token chain by applying one-way hash function on S_0.

Authentication:

In this step, the scheme generates the authentication token V at the client side. The authentication tokens are derived in the vertical chains by applying the one-way hash function on $Seed_1$. The authentication token is then attached to the transaction cookie and sent to the server. A similar authentication routine is done at the server's side to check the authenticity of the authentication token. If authentication is verified, the transaction is accepted; otherwise, it will be rejected. Once the first vertical chain is exhausted, the next routine (i.e. ***Seed Update***) is called to calculate seeds for the following vertical chain.

Seed Update:

Once the first vertical chain is exhausted, the seed is updated for the next chain by applying a one-way hash function on the initial S_0. Note that each seed is only used in a single vertical chain to generate authentication tokens for that chain. Furthermore, the authentication tokens once used are discarded and never used again. As such, the cost of authentication in the mini OHC scheme is a result of calculating the number of hash operations in the horizontal chain and the multiple vertical chains. The following is how we calculate the cost of the scheme:

$$\text{Cost of one vertical chain} = \frac{Y_i \times (Y_i+1)}{2}$$

$$\text{Total cost of all vertical chains} = C_V = X_i \times \frac{Y_i \times (Y_i+1)}{2}$$
$$= N \times \frac{(Y_i+1)}{2}$$

$$\text{Cost of the horizontal chain} = C_H = \frac{X_i \times (X_i+1)}{2}$$

$$\text{Total Cost} = C = C_V + C_H$$
$$= N \times \frac{(Y_i+1)}{2} + \frac{X_i \times (X_i+1)}{2}$$

B. The OHC Caching Scheme:

Unlike the previous mini OHC scheme, the OHC caching scheme utilizes storage and only one-dimension chain, to reduce the computation overhead of the OHC. During the initialization step, and in addition to the initial S_0 a few authentication tokens are pre-calculated and stored. Figure 2 demonstrates how the cache units are placed for this scheme. The highlighted blocks are where the authentication tokens are stored. For ease we assume the interval between caches is one. It should be noted that since we only have one dimension in the OHC caching scheme, the X parameter is considered the cache size (i.e. number of cache units utilized). Also, given we do not have vertical chains, we consider Y to be the interval between cache units. Therefore, X_j= size of cache Y_j= interval between two cache units.

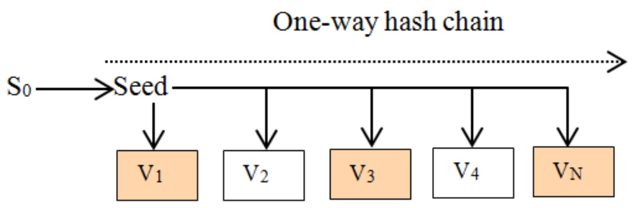

Figure 2 OHC Caching Scheme

Here is an example for a session of 100 transactions is authenticated using the OHC caching scheme. For a session of this size, five storage units (of length 160 bits for SHA-1) can be reserved. Thus five authentication tokens are calculated in the initial steps and stored at 20 transaction interval as follows. Note that X_j in this example is 5 and Y_j is 20.

cache[0] = s
cache[1] = H^{20}(s),
cache[2] = H^{40}(s),
cache[3] = H^{60}(s),
cache[4] = H^{80}(s).

Compared to the mini OHC scheme, the OHC with caching has the advantage of low computation cost. Two important parameters are used to guide the calculation of computation overhead in the OHC with caching: the cache size X_j and the cache spacing interval Y_j. In the above example, the cache size= 5 and the cache spacing interval=

20. Based on these assumptions, we can determine the cost of the OHC caching scheme according to the following formulas:

Cost of authentication tokens between two cache units $= \frac{Y_J \times (Y_J+1)}{2}$

$$\text{Total Cost} = C = X_j \times \frac{Y_J \times (Y_J+1)}{2}$$
$$= N \times \frac{(Y_J+1)}{2}$$

It should be noted that the total cost includes the sum of the cost of $(X_j\text{-}1) \times Y_j$ for the initial filling of the cache values and a cost of $\frac{(X_j-1) \times Y \times (X_J+1)}{2}$ for the N transactions. Also, notice that $(X_j\text{-}1)$ of the N transactions will not need to perform any hashing since the required value is already in the cache. For the above example of $N = 100$ and $X_j = 5$, transaction # 21 will simply read V_{80} from cache[4].

In order to handle more transactions efficiently, we either need to increase the number of storage units allocations. Or, we have to increase the cache spacing interval.

4 THE HYBRID SCHEME

We can alleviate the need for extra storage units and increase efficiency by equipping the mini OHC with sparse caching components to benefit from the advantages of caching in a two-dimensional configuration. Figure 3 is a general view of how the hybrid scheme looks with the caching units added to the mini OHC in the vertical chain Y_k. In the Simulation Results Section, we discuss why we prefer to equip the scheme with caching at the vertical chain. The highlighted blocks are where the sparse caching units are placed, and the authentication tokens in these locations are computed and stored. The proposed scheme uses two dimensions, each of which generates a set of values. In the first dimension—the horizontal dimension denoted X_k, is a single one-way hash chain responsible for generating seeds to be used in producing the authentication tokens in the second dimension Y_k—the vertical multiple one-way hash chains.

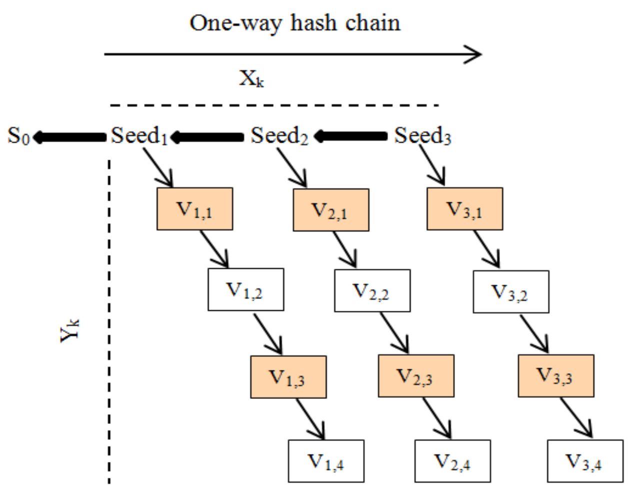

Figure 3 Hybrid Scheme

Here is a high level description of the protocol.

```
***Initialization***:
$X_k := N \div Y_k$                // length of the Seed_Chain
$K := Y_k$                         // K is the global index for the Token_Chain
$J := X_k$                         // J is the global index for the Seed_Chain
$Seed := H^J(S0)$                  // Seed is now $Seed_1$ for the first Token_Chain
Interval:= $Y_k \div Cache\_Size$  // # of hash operations between cache units
Call Fill_Cache(Seed)

***Fill_Cache (Seed)***
**Begin**
i:= 0
HNum:= 1                           // number of hashes to be calculated
While( i not equal to Cache_Size)
   Cache[i]:= $H^M(Seed)$          // authentication tokens stored
   HNum:= HNum +Interval
    i:= i+1
 End-While
**End**

***Authentication ( )***
**Begin**
L:= (K/Interval)-1                 //L is the cache locator to fetch the token value
HNum:= K-(L*interval)-1
$V := H^{HNum}(Cache[L])$
$K := K - 1$
   **if** ($K$==0) **then**
      Update_Seed( )
      $K := Y_k$;
   **end_if**
Return ($V$);
**End**

***Update_Seed( )***
**Begin**
$J := J\text{-}1$                  // J is the global index for the first-tier chain
$Seed := H^J(S0)$;
Call Fill_Cache(Seed)  // update the next authentication tokens
Return (Seed)
**End;**
```

The protocol is composed of four main procedures: the *Initialization*, the *Authentication*, the *Update_Seed* and the *Fill_Cache (Seed)* routines. Each of these routines is responsible for some part of the protocol.

The *Initialization* procedure works the same way as described in the mini OHC described in Section 3. An additional step in the *Initialization* entails filling the cache with authentication tokens based on the cache size. This is achieved by invoking the *Fill_Cache (Seed)* procedure.

The next step is when the session actually starts. It is where authentication tokens are used to protect session cookies. The *Authentication* procedure is responsible for generating the authentication tokens. This step works by locating the closest cache, fetching the respective stored authentication token and performing the additional hash operations if needed. Once the first vertical chain is exhausted, the *Update_Seed* step is invoked and a new seed is calculated and handed over to *Fill_Cache (Seed)* so that authentication tokens for the next Token Chain are stored. The protocol works in this manner until the session is complete.

Compared to the single dimension OHC caching scheme described in Section 3, where the number of cache units devised either grows proportionately with the number of transactions, or is configured to handle more transactions by increasing the cache spacing interval, the hybrid scheme is more efficient. In other words, to achieve good performance with higher number of transactions, the OHC caching scheme will need to devise more cache units. In the hybrid configuration, however, we efficiently handle this scenario, but with much less space by emptying storage after each Token Chain is exhausted.

While the mini OHC performance is influenced by the length of the Token Chain and the OHC with caching by the cache size, we need to investigate the optimal configuration of the hybrid scheme by comparing the costs of the previous two schemes and identifying the factors that influence the performance. In the following section, we introduce our evaluation of these factors and present an analytical model to find the best tradeoff between cache employment and performance.

5 COMPARISON AND TRADEOFFS

Essentially, using the number of hash operations in a session as a measurement metric, the mini OHC has higher computation cost as opposed to the OHC caching scheme. The difference between the two schemes is $\frac{X_i \times (X_i+1)}{2}$. However, there is the expense of extra storage units associated with OHC caching scheme. If the number of cache units devised is relatively small, the performance is comparable. However, if more cache units are added, the OHC caching scheme outperforms the mini OHC. Figure 4 shows how the two schemes give different performance for different size of cache for 500 transactions.

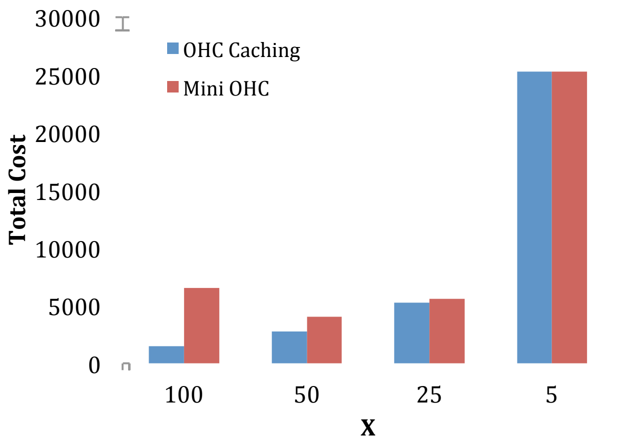

Figure 4 Total Cost of 500 transactions with different x values

In the hybrid scheme, our goal is to utilize the minimum storage requirements while efficiently handling authentication. First, we present how the total hash cost of the hybrid scheme is calculated. The analytical model below is used to obtain the optimal setup of the scheme; we try to achieve a configuration that strikes a balance between efficiency and memory requirements.

$$N = X_K \times Y_K$$

Assuming $X_K << Y_K$ and using the same number of sparse storage units X in the vertical chains, we get:

Let Space_Interval $M = \frac{Y_K}{X_K}$ // simplifying assumption M integer

$$Y_K = M \times X_K \quad N = X_K \times Y_K = M \times X^2$$

Cost of authentication tokens between 2 cache units $= \frac{M \times (M+1)}{2}$

Cost of one vertical chain $= X_K \times \frac{M \times (M+1)}{2}$

$$= Y_K \times \frac{(M+1)}{2}$$

Cost of all vertical chains $= C_V = X_K \times Y_K \times \frac{(M+1)}{2}$

$$= N \times \frac{(M+1)}{2}$$

Cost of horizontal chain $= C_H = \frac{X_K \times (X_K+1)}{2}$

Total Cost $= C = C_V + C_H$

$$= N \times \frac{(M+1)}{2} + \frac{X_K \times (X_K+1)}{2}$$

The above formula can be used to plot C as a function of N and M where $X = \sqrt{\frac{N}{M}}$

$$C = 0.5NM + 0.5N + \frac{0.5N}{M} + 0.5\sqrt{\frac{N}{M}}$$

To find the optimal value of M which minimizes the cost C, we differentiate the above formula with respect to M and equate to 0. On differentiation we have

$$\frac{dC}{dM} = 0.5\,N - 0.5\,\frac{N}{M^2} + 0.5\,\sqrt{N}\,\left(-0.5\,M^{-3/2}\right)$$
$$= 0.5N - 0.5\,\frac{N}{M^2} - 0.25\,\frac{1}{M}\sqrt{\frac{N}{M}}$$

After equating $\frac{dC}{dM}$ to zero, we get the following quartic equation:

$$4NM^4 - 8NM^2 - M + 4N = 0$$

To give an example, the Plot of the function C when the number of transactions is 500 is given below in Figure 5.

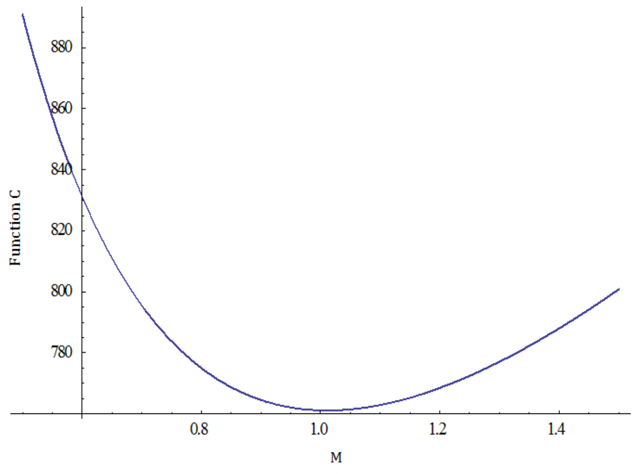

Figure 5 The minimum value of M when N=500

Thus, the minimum value occurs near the value M = 1. Table 1 summarizes the optimal values of M for the different numbers of transactions.

Table 1 Optimal values of *M*

N	Optimal value of M by equating $\frac{dC}{dM} = 0$
500	1.01118
1000	1.00791
1500	1.00645
2000	1.00559
2500	1.00500
3000	1.00456

Based on the analytical modeling presented above, we can determine the optimal cache spacing in the hybrid scheme to be 1. Therefore, given the optimal Token Chain length Y_k obtained in [15], we run our simulation with the assumption that the optimal cache spacing is 1. In the next section the simulation results for the three schemes are presented.

6 Simulation and Performance Results

The performance of the proposed hybrid scheme is evaluated using a detailed Java benchmark. Our goal was to measure the performance of the three schemes. We compared and contrasted the results measured in terms of efficiency (number of hash operations in a session) and in terms of storage units required to complete an internet session.

6.1 A. Caching Options:

The hybrid scheme can benefit from caching in a number of ways. Our first option is to use caching in the Token Chain Y_k. In other words, we only store the authentication tokens or a subset of them in the vertical dimension of the mini OHC. Caching can be either full or partial. In the full caching option, all the authentication tokens are calculated and stored before authentication, whereas only a subset of authentication tokens are calculated and stored in the partial caching option. Below is a description of both options.

In the full caching

The number of cache units required is equal to the optimal Token Chain length Y_k. Since all authentication tokens are going to be calculated and stored before the start of the session, the full caching approach indicates that the authentication tokens do not require any hash operation in the Y_k. The only cost incurred when full caching is utilized would be hash operations used to derive the seeds in the X_k.

In the partial caching

A subset of authentication tokens in the Token Chain Y_k is stored. We use the optimal cache spacing obtained above (i.e. M = 1) and the optimal Token Chain Y_k as the basis for our spacing. As a result, each authentication token will cost either one hash operation or none (i.e. fetching the authentication token form the cache). Here is an example to illustrate this:

Suppose we have an Internet session of length 500 transactions. According to [15], the optimal length of Y_k in the mini OHC is 10. The length of Seed Chain X_k is going to be 50.

Cache [0]=$H^1(S_1)$
Cache [1]=$H^3(S_1)$
Cache [2]=$H^5(S_1)$
Cache [3]=$H^7(S_1)$
Cache [4]=$H^9(S_1)$

Given the number of cache units and the spacing interval, the following are the first ten authentication tokens along with their cost in terms of hash operations.

1st Transaction = $V_1 = H^1$(Cache [4])
2nd Transaction = $V_2 = H^0$(Cache [4])
3rd Transaction = $V_3 = H^1$(Cache [3])
…….
10th Transaction = $V_{10} = H^0$(cache [0])

Therefore, for every Y_k only 5 hash operations are required given we have 5 cache units uniformly distributed. The maximum number of hash operations for the authenticate token transaction is 1 if partial caching with cache spacing of 1 is used.

These caches are used in the first Y_k. Once this chain is exhausted, the cache is emptied and a new seed is generated for the next Y_k authentication tokens. New values are calculated and the cache is filled again with new authentication tokens. If partial caching was used in the OHC with caching scheme only, we would need 250 memory spaces to carry out an Internet session of length 500 transactions to achieve comparable results. With adding partial sparse caching to the mini OHC, we can bring this number down to just 5 memory spaces.

6.2 B. Full vs. Partial Caching Performance

Figure 6 demonstrates the session cost measured by the number of hashes in the full caching configuration and the partial caching operation. We perform this test when caching is only performed at the Token Chain Y_k. Later, we test the caching option in the Seed Chain X_k. Here we can see that the full caching does not have a tremendous improvement over partial caching.

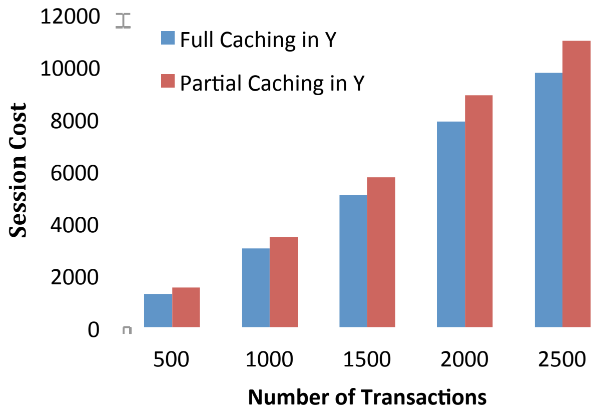

Figure 6 Session cost comparison between full and partial caching in Y.

Thus, the next step is to compare the storage requirements in the full and partial caching configurations in the Token Chain Y_k to see whether it is worth to employ full caching or partial caching. The comparison is presented in Figure 7. While the partial caching requires half the storage of the full caching, it can still achieve good results. Therefore, the partial caching can be a better option as the memory requirement is half without sacrificing performance.

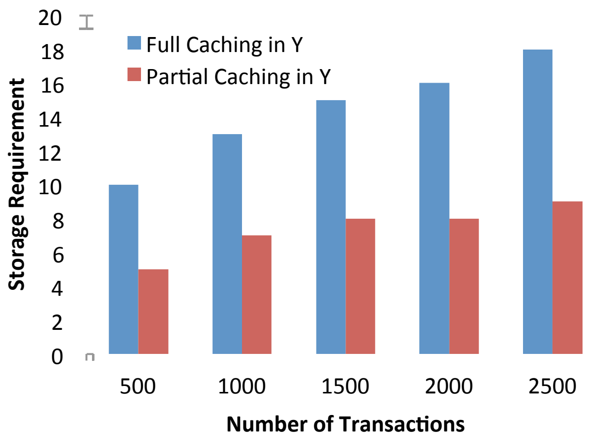

Figure 7 Storage requirement comparison between full and partial caching

6.3 Performance Comparison between the three schemes

Since the partial caching option strikes a good balance between memory requirement and efficient performance, we compare the performance of the hybrid scheme in the partial caching option at the Token Chain Y_k with the OHC caching scheme and the mini OHC scheme. Table 2 summarizes the results of this comparison. Note that we use the same number of cache units in the simulation of each scenario except for mini OHC where caching is not supported. In terms of efficiency, the mini OHC scheme helps reduce the session cost significantly if compared to the OHC scheme. By utilizing very little storage in the hybrid scheme, we were able to lower this cost by approximately 65% as indicated in the table.

Table 2 Comparing session cost between three schemes

Number of Transactions	OHC	mini OHC	Hybrid Scheme
500	25250	4025	1525
1000	71930	10009	3465
1500	141382	12505	5750
2000	251000	24875	8875
2500	348471	33496	10980

6.4 Caching in Token Chain Y_k or Seed Chain X_k

Our next task is to see whether equipping the proposed scheme with caching capabilities in both dimensions can have better outcome. Figure 8 and 9 demonstrate the storage and session cost requirements if either the Seed Chain or the Token Chain is equipped with caching capabilities. It is obvious that adding caching in the Seed Chain does not benefit the scheme as storage requirement increases (Figure 8) while the session cost increases (Figure 9). Therefore, we have opted for equipping the hybrid scheme with sparse caching at the Token Chain to be the optimal setup.

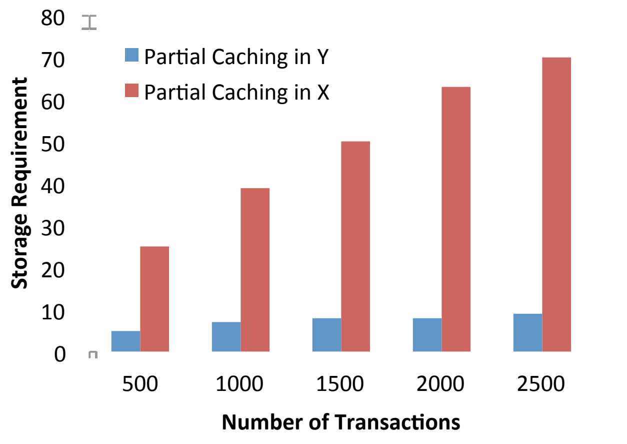

Figure 8 Storage requirement comparison with cache support in either Token_Chain (Y) or Seed_Chain (X)

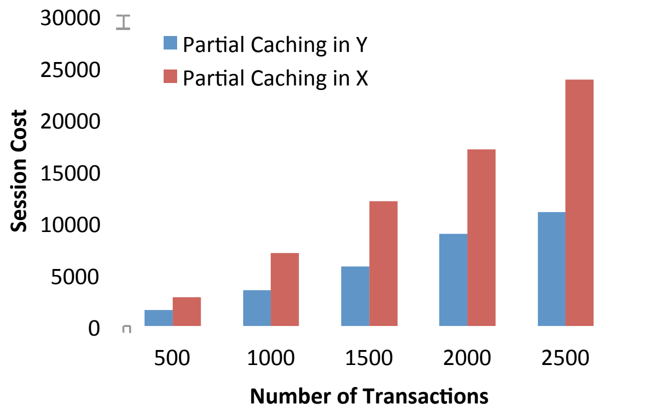

Figure 9 Session cost comparison with cache support in either Token_Chain (Y) or Seed_Chain (X)

7 Conclusion

This paper addresses the security threat of session hijacking attacks facing collaborative application especially when mobile and wireless applications are utilized to access collaborative services. Common HTTPS based solutions do not usually suit mobile devices especially those with limited computation and storage capacities. One-way hash chain based solutions have been proposed to replace the current cookie based session management techniques but due to their inherent nature requiring recursive computation of hash values, they do not suit some mobile devices. This is particularly because of the high computational overhead associated with OHC in Internet session.

This paper proposed and analyzed the potential of a hybrid solution where divided one-way hash chains are equipped with caching capacities to store pre-computed hashed values and fetch them once needed to authenticate a user session. We presented an analytical model which aimed at measuring the cots of the hybrid scheme compared to the straightforward OHC with caching and the two-dimensional OHC. We also used this analysis to derive a quartic equation with which we were able to identify the optimal cache spacing configuration in the hybrid scheme. The evaluation and experimentation reveal major improvements and highlight advantage of adding sparse caching to the mini one-way hash chains to achieve economic and efficient authentication for mobile devices that suits collaborative applications and other Internet applications.

References

[1] T. OWASP, "10 2010," *The Ten Most Critical Web Application Security Risks*, 2010.

[2] D. Wichers, "The 2013 OWASP Top 10," in *AppSec USA 2013*, 2013.

[3] D. Chrastil. (01/30/2014). *CookieCatcher - Session Hijacking Tool*. Available: http://security-sh3ll.blogspot.com/2013/08/cookiecatcher-session-hijacking-tool.html

[4] B. Ponurkiewicz, "FaceNiff—A new Android download application," ed, 2012.

[5] E. Butler, "FireSheep: Cookie Snatching Made Simple," in *ToorCon Conference, San Diego, CA*, 2010, pp. 22-24.

[6] S.-H. Wu, M.-J. Chou, C.-H. Tseng, Y.-J. Lee, and K.-T. Chen, "Detecting in-situ identity fraud on social network services: a case study on facebook," in *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, 2014, pp. 401-402.

[7] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai, "Uncovering social network sybils in the wild," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, p. 2, 2014.

[8] A. Mohaisen and S. Hollenbeck, "Improving Social Network-based Sybil Defenses by Rewiring and Augmenting Social Graphs," in *Information Security Applications*, ed: Springer, 2014, pp. 65-80.

[9] R. Gill, J. Smith, and A. Clark, "Experiences in passively detecting session hijacking attacks in IEEE 802.11 networks," in *Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54*, 2006, pp. 221-230.

[10] A. X. Liu, J. M. Kovacs, C.-T. Huang, and M. G. Gouda, "A secure cookie protocol," in *Computer Communications and Networks, 2005. ICCCN 2005. Proceedings. 14th International Conference on*, 2005, pp. 333-338.

[11] B. Adida, "Sessionlock: securing web sessions against eavesdropping," in *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 517-524.

[12] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen, "SessionShield: lightweight protection against session hijacking," in *Engineering Secure Software and Systems*, ed: Springer, 2011, pp. 87-100.

[13] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, pp. 770-772, 1981.

[14] I. Dacosta, S. Chakradeo, M. Ahamad, and P. Traynor, "One-time cookies: Preventing session hijacking attacks with disposable credentials," *ACM Transactions on Internet Technology (TOIT)*, vol. 12, 2012.

[15] J. Cashion and M. Bassiouni, "Protocol for mitigating the risk of hijacking social networking sites," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th IEEE International Conference on*, 2011, pp. 324-331.

[16] Amerah Alabrah, Jeffrey Cashion and Mostafa Bassiouni "Enhancing security of cookie-based sessions in mobile networks using sparse caching" International Journal of Information Security- Springer Publishing, Vol. 13, No. 4, July 2014, pp. 355–366, online version published December 2013, DOI: 10.1007/s10207-013-0223-8.

[17] A. Alabrah and M. Bassiouni, "A hierarchical two-tier one-way hash chain protocol for secure internet transactions," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, 2012, pp. 868-873.

[18] A. Alabrah and M. Bassiouni, "Robust and fast authentication of session cookies in collaborative and social media using position-indexed hashing," in *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th IEEE International Conference Conference on*, 2013, pp. 241-249.