DoubleCheck: Multi-path Verification Against Man-in-the-Middle Attacks

Mansoor Alicherry Angelos D. Keromytis
Department of Computer Science, Columbia University in the City of New York

***Abstract*— Self-signed certificates for SSL and self-generated hosts keys for SSH are popular zero-cost, simple alternatives to public key infrastructure (PKI). They provide security against man-in-the-middle attacks, as long as the the client connecting to those services knows the certificates or host keys *a priori*. A simple solution used in practice is to trust the certificate or the host key when the client connects to a server for the first time. This approach is susceptible to man-in-the-middle attacks, a fact exploited by adversaries in a variety of attacks against unsuspecting users. We develop a simple and scalable solution named *DoubleCheck* to protect against such attacks. Our solution is achieved by retrieving the certificate from a remote host using multiple alternate paths. Our scheme does not require any new infrastructure; we make use of the Tor anonymity system to reach the destination using multiple independent paths. Hence our solution is easy to deploy in practice. Our solution does not introduce any privacy concerns. We have implemented DoubleCheck as SSH and Firefox extensions, demonstrating its practicality. Our experimental evaluation shows that the impact of DoubleCheck on performance is minimal, since the Tor network is used only for retrieving the certificate for the first time, while the data transfer and subsequent connection establishment follow normal routing rules. Our scheme is an effective way of mitigating the impact of man-in-the-middle attacks without requiring new infrastructure and at low overhead.**

Keywords: Man-in-the-middle attack, Certificates, Trust, Tor

I. Introduction

SSL and SSH provide secure alternatives to HTTP and telnet/rlogin by associating certificates or host keys to network servers. The protocols verify the identity of the remote servers using those certificates or host keys. The security of these protocols depends on the robustness of the verification of this identity.

SSL typically uses public key infrastructure (PKI) to verify the identity of the remote server. This requires network servers to obtain certificates from a well-trusted certificate authority (CA), such as VeriSign. These CAs issue a certificate to an organization only after conducting background checks on its identity. Though this PKI infrastructure provides high level of confidence on the identity of the servers carrying those certificates, obtaining a certificate is very expensive and time-consuming for an organization. For example, a SSL certificate with one year validity costs $399 and $995 with 40-bit and 128-bit minimum encryption respectively [8]. The proof of identity required by the CA includes documentation filed with a government agency or a competent authority, or verification with a third party identity proofing service [7].

A simple and zero-cost alternative to PKI certificates is the use of self-signed certificates. In that scheme, a server issues a certificate for itself. Though this resolves the problem of possessing a certificate for the server, a client connecting to that server is faced with the problem of having to accept a certificate that it does not trust. Whenever a client application (*e.g.*, web browser) connects to such a server, the user is prompted whether he/she accepts its certificate. If the user accepts the certificate, it is added to the list of trusted certificates. Any future connection to that server is verified using this list. A similar approach applies to SSH, which is a secure alternative to telnet and rlogin. Instead of certificates, each host generates a host key that acts as the identity of that host. There is no well-known trusted third party to certify that identity. A user is prompted for accepting the host key, whenever the SSH client connects to a new server. If the user accepts the key, it is cached in the list of trusted keys.

Even though self-signed certificates and host keys provide a scalable and inexpensive solution, security depends on the user being able to "judge" if the certificates are valid. Since there is no automated, easy-to-use way to know if the certificate belongs to the right host, he/she typically accepts the certificate whenever prompted. Studies have shown that users were willing to transact online based on visual appearance and professionalism of the web site, and ignore the security warnings of the web browser [18], [16]. As this happens whenever the user connects to a server for the first time, this paradigm is called *trust on first use (tofu)*.

Though trust on first use system has seen widespread use, it has two major security shortcomings:

1) **Man in the middle attacks:** *tofu* cannot protect against man-in-the-middle (MitM) attacks when a client is connecting to a server for the first time, since the client accepts the attacker's identity. Any future connection to the correct server will be declared as a man-in-the middle attack.
2) **Changing the server certificates:** Whenever a server changes its certificate, existing clients must determine whether a MitM attack is underway or whether the server legitimately changed its certificate/host key. Since the latter happens more frequently than the former, users are likely to opt for the benign explanation even when an actual attack is underway.

The recently proposed *Perspectives* [20] system addresses the problem of needing to accept the unknown certificate or

host key by introducing a set of servers on the Internet called *notary servers*. Notary servers maintain a database of the hash of the identities of the Internet servers over a period of time. A client program connecting to a remote server whose certificate is not trusted gets the history of the certificate from multiple notary servers and makes a decision about accepting or rejecting the certificate. Though Perspectives is very robust for verifying the identity of the remote servers, its effectiveness depends on the universal availability of multiple notary servers. The notary servers need to be powerful in terms of processing power, memory and bandwidth as they need to probe for and keep track of the keys/certificates of all the Internet servers. They also need to update the certificate information of all those servers periodically to address the problem of changing the server certificate mentioned above.

In this paper, we address the two problems of "trust on first use" by retrieving the certificates or host keys using alternate paths. We call our solution *DoubleCheck* as we accept a connection with a remote server only if the certificate retrieved on various paths match. The certificate retrieval on the alternate paths is done on-demand using the existing and widely available infrastructure of Tor network [5]. Hence we get most of the benefits of Perspectives without the need of new infrastructure. Since the alternate path certificate retrieval is done via anonymity network, our solution does not introduce any privacy concerns.

The primary contributions of the paper are the following:

- We present *DoubleCheck*, a practical scheme for validating the certificates and host keys by making use of the existing Tor infrastructure.
- We offer the implementation of DoubleCheck for both SSL and SSH protocols that can be easily deployed on end user systems.
- We provide a performance analysis of DoubleCheck using implementation, which encounters an overhead only when a server with a key that is not trusted is accessed for the first time.

II. Threat model

Our solution is designed for an adversarial Internet environment, where clients connect to servers whose identity is not known to the client in advance using SSH or HTTP/SSL[1]. An adversary may masquerade as the server that the client is trying to connect to, without directly compromising the server. Our goal is to protect the clients from connecting to such adversaries. We do not protect against clients connecting to legitimate but compromised servers. We also do not protect against attacks where client connects with a fake server controlled by an attacker, like phishing attacks.

An adversary can launch a man-in-the-middle (MitM) attack by redirecting all the traffic from a client destined to a server to it. This can be done through DNS spoofing, by route manipulation, or by direct access to a link on the path between the client and the server [6], [13]. The attack can be launched from various points:

1) **The attacker launches the attack closer to the clients:** This can be done if the attacker is present in the same subnet as the client. This is easy to launch, especially if the attacker is providing a "free service" to the client, like unsecured WiFi access, or is using the same unsecure network (*e.g.*, public or unencrypted WiFi). Only the clients in that subnet are affected by this attack.
2) **The attacker launches the attack closer to the server:** Here the attacker is part of the server subnet or has compromised the first-hop router of the server or the DNS server. This kind of attack is typically harder to launch unless the attacker has access to the server subnet. All the clients connecting to that server are affected by this kind of attack.
3) **The attacker launches the attack from the middle:** This is done by compromising a router in the path between the client and the server. This is also generally harder (but not impossible) to launch. Only the clients whose path to the server uses the compromised router are affected by this attack. Another way to launch this attack is by compromising one or more DNS servers.

Our solution can protect the clients against the identity attacks when the attacker is not in the subnet of the server (Case 2, above). An attacker can launch an attack only if the server's identity is not known to the client in advance.

III. System Architecture

The basic idea of DoubleCheck is to fetch the certificate or host key from a remote server using at least two different paths. If the certificates received are the same, then with high confidence the identity of the remote server can be trusted. DoubleCheck cannot protect against attacks where the attacker is part of every path to the remote server. For example, DoubleCheck cannot detect an attack that is taking place at the default gateway of the subnet where the remote server is located.

A. Overview of Tor

Tor[2] is a network of virtual tunnels that allows users to improve their privacy and security on the Internet [5]. Instead of taking a direct route from source to destination, data packets on the Tor network take a random pathway through several relays such that no observer at any single point can tell where the data came from or where it's going. To create a private network pathway with Tor, the client software incrementally builds a circuit of encrypted connections through relays on the network. The circuit is extended one hop at a time, and each relay along the way knows only which relay gave it data and which relay it is giving data to. No individual relay ever knows the complete path that a data packet has taken. The client negotiates a separate set of encryption keys for each

[1]Our scheme is compatible with other public key cryptography-based protocols, including IPsec.

[2]An acronym that stands for "The Onion Router".

hop along the circuit to ensure that each hop can't trace these connections as they pass through.

Our primary motivation for using Tor is not for its anonymity, but for its ability to to create multiple alternate paths to a remote server. Users of the Tor network run a local Tor proxy, which connects to the Tor network. This proxy also exposes a SOCKS [14] interface for that host's applications to connect. Traffic inside the Tor network goes through multiple relay nodes encrypted; ultimately reaching an *exit node* that forwards the traffic to its real destination. The destination node sees the connection as coming from the exit node. We can create multiple paths to a destination by connecting through the Tor network and configuring it to use different exit nodes.

The advantage of using Tor is that it is a well established, widely used and stable. There are more than 1000 Tor relay nodes as of January 2009 [4]. So we do not have build it from scratch, which makes our scheme very easy to deploy in practice.

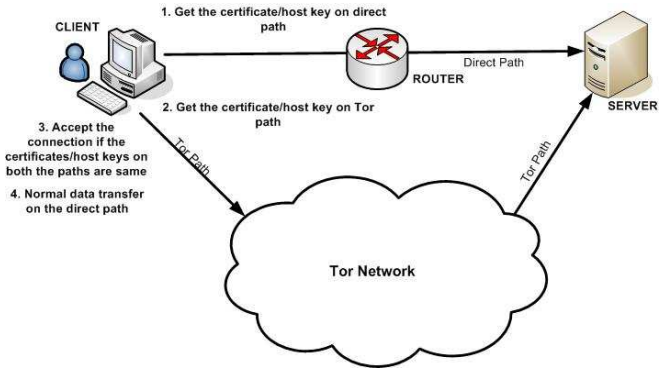

Fig. 1. DoubleCheck system overview

B. DoubleCheck

The details of DoubleCheck are depicted in Figure 1. When a user wants to connect to a remote server, the client application fetches the remote host certificate on the normal (direct) path. If the certificate is already trusted by the client, the connection is accepted and communication takes place as normal. If the certificate is not already trusted by the client, because of either connecting to a new server or due to a change in the certificate at the server, another connection to the server is attempted using the Tor network. This connection is established by the client application by connecting to the local Tor proxy. The Tor proxy in turn connects to the destination using the Tor network as described above. This will result in the client application establishing a connection to the remote server via an alternate path. The remote server's certificate is fetched from the remote server by the client application using this new connection. The client application then compares the certificate it received on the normal path with that received over Tor. If they are different, the connection is aborted as there is a possibility of a MitM attack. If the certificates are the same, then the client could either accept the connection, or try a second alternate path through the Tor network by changing the exit node. This second alternate path increases the confidence in the validity of the certificate, at the expense of higher overhead. Once the client application is satisfied about the identity of the remote server, the connection goes on the normal path and the certificate is added to the list of trusted certificates, for the future use depending on the policy. A client may initiate multiple simultaneous alternate paths to achieve higher confidence while minimizing connection latency.

C. Security analysis

We are focusing on man-in-the-middle attacks on the servers that do not use a well trusted PKI mechanism. These include self signed SSL certificates and self generated SSH keys. A well trusted PKI for SSL or out of band verification of SSH keys provide highest security against man-in-the middle attacks.

The effectiveness of DoubleCheck depends on finding a path to the destination node that does not contain the attacker. Doublecheck will fail to detect the attacker either if all paths to the destination pass through the attacker or if all the paths tried by it passed through the attacker. The former can be handled only by a solution that keeps track of the certificate history. The later can be avoided in DoubleCheck by trying multiple alternate paths by intelligently choosing the exit nodes.

Here we analyze DoubleCheck against the threat model discussed in Section II. If the server has a certificate that is trusted by the client, then the client does not retrieve the certificate in the alternate path. This can happen if the certificate is stored in the local cache of the client or the certificate is signed by a trusted CA. A client using a trusted certificate might end up connecting to an attacker only if the server is compromised. We are not protecting against clients connecting to compromised servers.

An attacker can launch a man-in-the-middle attack by making sure that all the packets destined to a server from a client reach the attacker. This can be achieved either by making clients believe that the IP address corresponding to the server's name is the IP address of the attacker (by spoofing the DNS) or by taking over the server's IP address (*e.g.*, by manipulating the BGP routing table). As discussed in the threat model, it is easier to launch an attack if the attacker is closer to the client (e.g. attacker is in the client subnet). In those cases the effectiveness of DoubleCheck is also high, since it is much easier to find an alternate path to destination that does not contain the attacker using the Tor network. In fact it is highly likely that any Tor path will bypass such an attacker. It is important that the domain name of the server be resolved by the Tor network while fetching the certificate in the alternate path, rather than resolving the name locally, to protect against large-impact DNS poisoning attacks. But, if the authoritative DNS name server for the domain is compromised, the even Tor name resolution will also get redirected to the attacker's choice of IP address.

If the attack is not launched from the client subnet, it is possible that the attacker is on the path to the server from both the client and the Tor exit node. This is true if the attacker is

on the server subnet, and DoubleCheck cannot detect it. A Tor exit node itself can be under attack. The attacker could have control over the Tor exit node, which could send the attacker's certificate to the client. A client can reduce the probability of such an occurrence by getting the server certificates using multiple Tor circuits by changing the exit nodes or by using multiple simultaneous Tor paths with different exit nodes.

IV. Implementation

We implemented DoubleCheck scheme for both SSH and SSL. The design philosophy for both the protocols is the same. Whenever the client (`ssh`/`scp` command or the browser) connects to a server, it acquires the server's host key on the direct path. If the host key is suspicious or previously unknown, we use the anonymous (Tor) network to retrieve the host key from the same server. If the host keys are the same then the connection is allowed on the direct path. The Tor network is used only for retrieving the keys and not for the actual data transfer. If there is a key mismatch, the connection is aborted.

We make use of the Tor client, without any modification, to retrieve the host keys on the alternate path. Tor client acts as a SOCKS proxy that listens on port 9050 and forwards connections to the Tor network.

A. SSH Implementation

Secure shell or SSH is a network protocol that allows data to be exchanged between computers securely. SSH uses public key cryptography for authentication. The `ssh` command in Unix is used for executing commands on the remote computer using the SSH protocol. To verify the identity of the remote computer, `ssh` maintains a list of trusted computers and their keys in a file (typically, `~/.ssh/known_hosts`). When a user connects to a remote server that is not on the list of trusted computers, `ssh` displays the fingerprint of the key received, and prompts the user whether he wants to add that computer and key to the trusted store. To verify the the fingerprint, the user need to resort to an out of band mechanism like a fax, a phone or an email. There is no automated, easy-to-use way for the user to know if the host key of the remote computer is correct, and hence it is accepted by the user on a presumption of validity (Trust on First Use). Any connection to that server from that point onwards is verified by the `ssh` client by comparing the server's key with the one stored in the trusted store. This authentication method closes security holes due to IP spoofing, DNS spoofing, and routing spoofing after the first connection.

The `ssh` command from openSSH [2] provides a rich set of functionality, which enabled us to implement DoubleCheck with a simple shell script. The following command can be used to retrieve the key of a remote server ($SERVER) and store it in a temporary file ($KEY-FILE-DIRECT) using the direct path.

```
ssh $SERVER -o StrictHostKeyChecking=no
-o UserKnownHostsFile=$KEY-FILE-DIRECT
-n -N -o NumberOfPasswordPrompts=0 -o
PasswordAuthentication=yes
```

To retrieve the host key using Tor, we run the Tor client (started as a daemon process) and make use of the *netcat* (nc) command. The Tor client runs SOCKS proxy on port 9050. Netcat provides a rich set of functionality to connect using UDP or TCP. It can be used for connecting using an HTTPS or SOCKS proxy, and can be used as a proxy option of the `ssh` command. Specifically, the *ProxyCommand* option of `ssh` uses the command given as its argument to establish the network communication. The following command can be used to retrieve the key of a remote server ($SERVER) and store it in a temporary file ($KEY-FILE-TOR) using the Tor network.

```
ssh $SERVER -o StrictHostKeyChecking=no
-o UserKnownHostsFile=$KEY-FILE-TOR
-n -N -o NumberOfPasswordPrompts=0
-o PasswordAuthentication=yes -o
ProxyCommand='/usr/bin/nc -X 5 -x
127.0.0.1:9050 %h %p'
```

Now we can compare the contents of the host keys retrieved on the direct path and the Tor path. If they are different, then there is a likely MitM attack for the remote server. If the host keys are the same, the attack is unlikely as described in Section III. If we want to trust the cached host keys that are present in the `.ssh/known_hosts` file, we execute the following command before we retrieve the host keys on the direct and Tor paths.

```
ssh $SERVER -o StrictHostKeyChecking=yes
```

This command succeeds only if the host signature is available on the cache and matches with the one retrieved on the direct path.

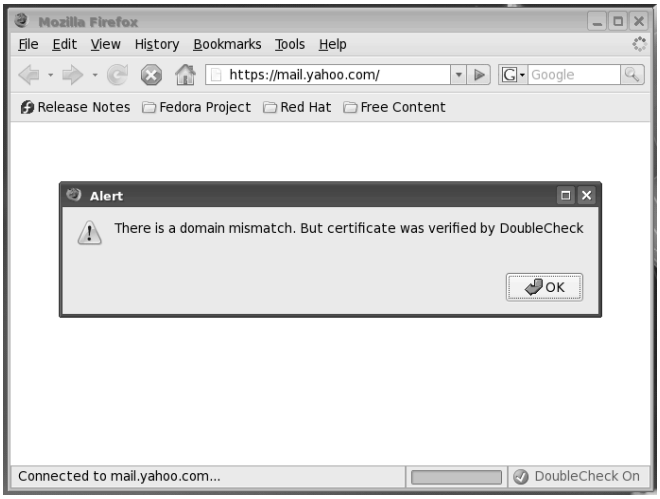

Fig. 2. User alert on domain mismatch

B. SSL Implementation

SSL is a cryptographic protocol that provides secure communication on the Internet for web browsing and other data transfers. Its most common use is for secure browsing using

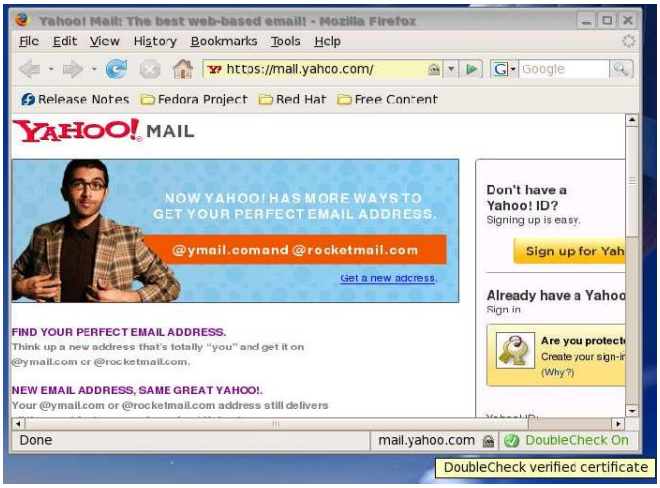

Fig. 3. Secure browsing using DoubleCheck

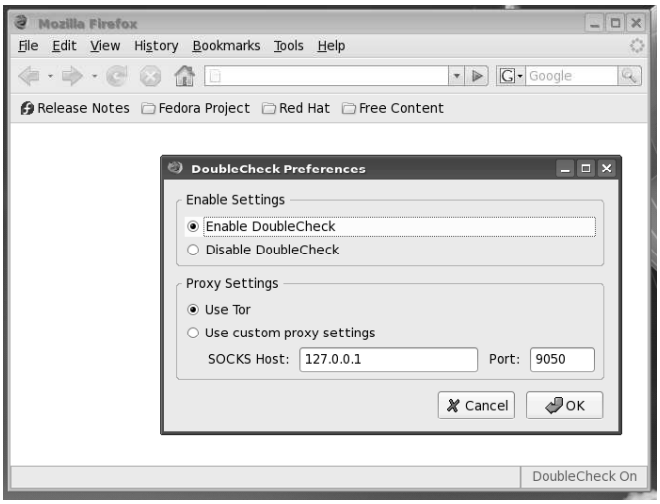

Fig. 4. Preference settings for DoubleCheck plugin

HTTPS (HTTP over SSL). During the connection establishment of SSL, the server sends its certificate to the client. The verification of the certification by the client typically involves the following:

1) The server certificate is signed by one of the certificate authorities (CA) that the client trusts.
2) The server certificate has a valid time period.
3) The named owner in the certificate is the same as the server to which the client connected to.

Web browsers warn the user if the user attempts a connection to a site that fails any one of the above checks. The user is prompted for accepting the violating certificate for the current session or permanently. Though the user has more information about the certificate compared to the SSH host keys case (such as the signer, whom the certificate is issued to, expiration, *etc.*), he/she still would not know if the certificate can be trusted because of the violations. Typically, users grant permission to connect temporarily to the server. The DoubleCheck mechanism can provide information to the user that enables an informed decision to be made.

We implemented a DoubleCheck plugin for the Firefox browser. The plugin blocks access to sites for which the DoubleCheck reports possible attacks. The plugin is easy to use and can be enabled or disabled by click of a button. When enabled, it gives the status of the connection. Screenshots of the plugin in action are shown in Figures 2 and 3.

We implemented the Firefox plugin by first creating an overlay extension to the status bar that is loaded at the Firefox startup. We also created a DoubleCheck XPCOM [9] component that is invoked when any of the certificate validation checks fail. The browser is made to invoke the new XPCOM component functions by replacing the *@mozilla.org/nsBadCertListener;1* component with the new one. When invoked, the new XPCOM component retrieves a certificate on an alternate path over Tor and compares it with the one received by Firefox on the direct path. To receive the certificate on the alternate path, the XPCOM component uses OpenSSL [3] to connect to the remote server. This OpenSSL connection uses the Tor client as the SOCKS proxy, forcing the connection to go through the Tor network.

If the certificate retrieved by the XPCOM component on the alternate path is the same as the certificate retrieved by the Firefox browser on the direct path, the connection is allowed to go through. Otherwise, the connection is blocked. In either case, the result of the certificate retrieval is communicated to the status bar extension, which is then displayed as a tool tip on the status bar.

The Firefox plugin provides a preference window that allows the user to enable or disable DoubleCheck. The preference window also allows to user to run the Tor proxy on a non-standard port or on a remote machine (Figure 4). For security purposes, it is recommended that the user runs the Tor proxy on the local machine. The DoubleCheck Firefox extension is available at `http://www.cs.columbia.edu/~mansoor/doublecheck/`

V. Experimental Results

A. SSL performance

To determine the impact of DoubleCheck on regular browsing sessions, we measured the time it takes to load pages that have domain mismatch errors, as well as those with self-signed, previously unknown certificates.

For testing domain mismatch certificates, we used sites that redirect the other URLs whose domain does not match the information in the certificate. For example, `https://mail.yahoo.com` uses a certificate issued for `login.yahoo.com`, and `https://gmail.com` uses a certificated issued to `mail.google.com`. For self-signed certificates, we have used our own web servers configured appropriately.

The results of loading pages are shown in Table I. We use the notation *dc* for an experiment with DoubleCheck enabled and *org* for an experiment with DoubleCheck disabled. Each experiment was run 6 to 10 times and the average time was taken.

Error type	URL	experiment	time (s)
Domain mismatch	mail.yahoo.com	dc - bootup	13.847
		dc - first request	7.203
		org - first request	2.700
		dc - page refresh	0.933
		org - page refresh	0.889
Domain mismatch	gmail.com	dc - bootup	4.762
		dc - first request	3.200
		org - first request	2.700
		dc - page refresh	0.388
		org - page refresh	0.389
Self signed	private site 1 (IP address access)	dc - bootup	9.406
		dc - first request	6.059
		org - first request	4.859
		dc - page refresh	0.185
		org - page refresh	0.184
Self signed	private site 2 (DNS access)	dc - bootup	17.733
		dc - first request	14.664
		org - first request	11.636
		dc - page refresh	0.512
		org - page refresh	0.519
No error	login.yahoo.com	dc - first request	0.846
		org - first request	0.830
		dc - page refresh	0.546
		org - page refresh	0.508

TABLE I

TIME IN SECONDS TAKEN TO LOAD HTTPS PAGES

Server Location	Direct Path	Tor Path
Same subnet	0.140	5.243
Same campus	0.910	4.516
Same campus	0.942	10.287
Remote	0.189	5.117
Remote	0.871	14.839
Remote	3.287	9.105
Remote	0.423	13.887

TABLE II

TIME IN SECONDS TO RETRIEVE SSH HOST KEYS

When there is a domain mismatch or a self-signed certificate, DoubleCheck fetches the server certificate using the Tor network. This incurs an additional overhead compared to a system that does not use DoubleCheck. This additional overhead is higher when a page is fetched for the first time after a boot or after a period of system inactivity. This is due to the fact that the Tor client needs to re-establish the connection with Tor network. Hence the time to load a new site is divided into two rows for the DoubleCheck scheme in the table, marked as bootup and first request. The first time page loading (bootup) takes about 3.69 seconds longer.

We then measured the time taken to fetch a page, where the browser is restarted for each experiment. When DoubleCheck is disabled and there is a domain mismatch, the user is notified of the domain mismatch and prompted to press OK. If DoubleCheck is enabled, there is no prompt; DoubleCheck fetches the certificate through the Tor network and compares it with the one fetched through the direct path. The time taken to fetch the page for the DoubleCheck scheme is about 2.26 seconds longer compared to the original (insecure) scheme.

Once a page is loaded on to the browser, subsequent refreshing of the page or visiting other pages on the same site is faster than the initial loading. Navigating through the pages from a site is the most common operation of a user. There is no special processing needed for DoubleCheck to do this frequent operation, and hence there is no additional overhead.

The last set of rows on the table show the results for HTTPS sessions that do not have any issues with the certificates. In this common case there is no additional overhead for DoubleCheck.

B. SSH performance

Now we study the performance of the implementation of DoubleCheck for SSH using the scripts outlined in Section IV-A. We measure the time taken to retrieve the host keys on the direct and Tor paths. The experiments were conducted using the servers on same subnet as the client, same campus but different subnet, and different remote locations. Table II gives the time in seconds taken to retrieve the host keys. The additional time taken to retrieve the host keys on Tor path varied from 4 seconds to 14 seconds on average. There was no correlation on the time it took to retrieve the host keys and the location of the server. The time might depend on the actual path taken, and the load and capacity on the server as well as the Tor relay nodes. This additional time is needed only when connecting to a server for the first time, or when there is a key mismatch. Subsequent traffic and connections goes over the direct path, and there is no overhead. We believe that this additional time is not a high price to pay for the security.

VI. RELATED WORK

Perspectives [20] addresses the problem of verification of a server's identity through multiple path probing by creating a new infrastructure comprised of "notary servers." Notary servers probe the keys of all Internet-connected servers periodically and store the hash of the keys/certificates in a database. Clients request the history of keys from the notaries while connecting to an Internet server.

There are various solutions proposed for authenticating Internet hosts. X.509 certificates in PKI [12] and the SSH model are the two most popular. Other solutions include Kerberos [19] and web-of-trust approaches such as Pretty Good Privacy (PGP) [10]. Self organized public key management systems were proposed for Mobile Ad-hoc Networks (MANETs), where on-line access to trusted authorities are not available, by users creating their own certificates and accepting another certificate if there is a chain of trusted certificates [11].

RFC 4255 [17] proposes to use DNS to securely publish SSH host key fingerprints. This requires the deployment of secure DNS, as well as for DNS to act as a certificate authority for the host key fingerprints for the machines in its domain.

There has been considerable work to secure the DNS lookup process, which can prevent man-in-the-middle attacks through DNS spoofing. ConfiDNS [15] improves the security by multi-site lookup and lookup histories. The Domain Name System Security Extensions (DNSSEC) [1] add data origin authentication and data integrity to the Domain Name System.

	DoubleCheck	Perspectives
Spatial verification	Yes	Yes
Temporal verification	No	**Yes**
Multiple witnesses for keys	Yes	Yes
Number of witness	**Many**	4 (on 01/2009)
New infrastructure	**Not needed**	Needed
Methodology	On demand	Offline retrieval
New server or key change	**Supported**	Not supported
Privacy concerns	**No**	Yes

TABLE III
COMPARISON OF DOUBLECHECK AND PERSPECTIVES

A. Comparison with Perspectives

In this section, we compare and contrast DoubleCheck with Perspectives. DoubleCheck performs a subset of the functions Perspectives supports, but without requiring any new infrastructure. Table III summarizes the key differences. The scheme that has a clear advantage over the other is marked in bold.

The major advantage of Perspectives over DoubleCheck is that the former can support temporal verification of the certificates since the notary servers store the history of the certificates. Unlike DoubleCheck, this can detect a compromised server or an attack where the attacker is in the same subnet as the server. The disadvantage of temporal verification is that it can lead to false positives when the key of the server changes for genuine reasons.

The major advantage of DoubleCheck over Perspectives is that DoubleCheck runs over the existing infrastructure. Unlike Perspectives, no new powerful servers connected to the Internet via high bandwidth links are needed. Tor is well established and runs on large number of servers (1072 routers as of January 2009 [4]). Hence DoubleCheck is easier to deploy in practice. There can be privacy concerns in Perspectives as the clients need to reveal to the notary servers the address of the servers they want to connect to. Unless the notary servers do an on-demand retrieval of the certificates, a client using Perspectives will not be able to validate the keys for new servers unknown to the notaries.

VII. CONCLUSIONS

We presented DoubleCheck, a solution for verifying the certificates and host keys for "trust on first use" applications. Unlike previous proposals, our scheme is practical, easy to deploy, without any privacy concerns and does not require any new infrastructure. We showed that the solution is simple to implement and can be achieved by a straightforward script-based extension to SSH and an easy-to-use plugin for Firefox. We showed that performance overhead is minimal, with DoubleCheck incurring no overhead in the most common usage scenarios. Our scheme can mitigate the impact of man-in-the-middle attacks at low overhead without requiring any additional infrastructure.

ACKNOWLEDGEMENTS

This work was supported by NSF Grant CNS-07-14277 and ONR MURI Grant N00014-07-1-0907, with additional support from Google. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, ONR, or the U.S. Government.

Mansoor Alicherry is supported by Alcatel-Lucent, Murray Hill, NJ.

REFERENCES

[1] DNSSEC: DNS Security Extensions Securing the Domain Name System. *http://www.dnssec.net/*.
[2] OpenSSH. *http://www.openssh.org/*.
[3] OpenSSL. *http://www.openssl.org/*.
[4] Tor Network Status. *http://torstatus.blutmagie.de.*
[5] Tor project. *http://www.torproject.org*.
[6] US-CERT Vulnerability Note. Multiple DNS implementations vulnerable to cache poisoning. *http://www.kb.cert.org/vuls/id/800113*.
[7] VeriSign Certification Practice Statement. *http://www.verisign.com/repository/CPS/*.
[8] VeriSign SSL Certificates. *http://www.verisign.com/ssl/index.html*.
[9] XPCOM. *http://www.mozilla.org/projects/xpcom/*.
[10] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. OpenPGP Message Format. *RFC 4880*, Nov 2007.
[11] S. Capkun, L. Buttyn, and J. pierre Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2:52–64, 2003.
[12] S. Chokhani and W. Ford. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. *RFC 2527*, March 1999.
[13] D. Kaminsky. Black Ops 2008 – Its The End Of The Cache As We Know It. *Blackhat 2008 briefings. http://www.doxpara.com/DMK_BO2K8.ppt*.
[14] D. Koblas and M. R. Koblas. SOCKS. *USENIX Security Symposium*, 1992.
[15] L. Poole and V. S. Pai. ConfiDNS: leveraging scale and history to improve DNS security. *USENIX Workshop on Real, Large Distributed Systems*, 2006.
[16] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. Emperors new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. 2007.
[17] J. Schlyter and W. Griffin. Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints. *RFC 4255*, Jan 2006.
[18] J. Sobey, R. Biddle, P. van Oorschot, and A. Patrick. Exploring user reactions to browser cues for extended validation certificates. *European Symposium on Research in Computer Security*, October 2008.
[19] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. *Winter 1988 Usenix Conference*, February 1988.
[20] D. Wendlandt, D. Andersen, and A. Perrig. Perspectives : Improving SSH-style Host Authentication with Multi-path Network Probing. *USENIX Annual Technical Conference*, 2008.