OTIP: One Time IP Address

Renzo Davoli
Computer Science and Engineering Department
University of Bologna
Bologna, Italy
Email: renzo.davoli@unibo.it

***Abstract*—One Time IP address (OTIP) is a security feature to protect private communications on the Internet. OTIP enables nodes to change their IP addresses periodically following a cryptographic sequence. Legitimate users have all the information needed to compute the current addresses used by the servers, thus their networking clients are able to access the required services. OTIP current address is the output of a computation based on the Fully Qualified Domain Name (FQDN) of the server, a secret password and the current time. The major achievement of OTIP is that all the IP addresses collected by wiretapping the networks are useless for attackers as, in a short time, all the servers will be using different addresses.**

***Index Terms*—IP networks; TCPIP; Information security.**

I. Introduction

OTIP means that the current IP address of a server changes periodically to prevent networking attacks. The current IP address of a server is computed on the basis of some private information shared by legitimate users and the server itself, like a password, and the current time. This method has mainly been designed for IPv6 networks. In fact, the current server address can be picked up as one of the valid host addresses available on the local network, most of the time among 2^{64} possible addresses. Clearly, a 2^{64} address space is too large for attackers to try a brute force enumeration attack on all the available addresses; even if they eventually succeeded, the retrieved addresses would have to be exploited before their validity expires and the servers move to new addresses.

In theory, the same method could also be applied to IPv4, but it would be ineffective, due to scarcity of addresses and to the narrowness of address spaces within a network, most of the times 256 nodes or less.

The paper is organized as follows: the next section discusses the proposal, while section III compares OTIP with other methods known in the literature. Section IV, which gives an estimation of the probability that two servers may temporarily choose the same address, is followed by the implementation section, which describes the experimental part of the paper. The final section is about the limits, future developments and conclusive remarks.

II. Discussion

The Internet supports services for the general public as well as private services for a predefined set of authorized users. For a business firm or a company, its institutional web site is generally a service provided for the general public. On the other hand, a remote shell service for system administration or a Voice over IP (VOIP) service interconnecting the company's (software) Private Branch Exchanges (PBX) are examples of private services.

All the private servers clearly have a means of protection to prevent access by unauthorized users (e.g., password protections and traffic encryption). OTIP aims to provide one further layer of protection for private services.

Without OTIP, attackers can collect the IP addresses of the servers by wiretapping the network and creating a catalog of valid addresses and services. These addresses can later be used to perform brute force attacks, for instance using a database of weak passwords, or even to test vulnerability by using a collection of well-known exploits of the servers' code.

OTIP can prevent these attacks, or at least makes them extremely hard to succeed, as the addresses collected by network sniffers expire in a short time.

OTIP does require the Real Time Clocks (RTC) of servers and clients to be synchronized, as the current time is a parameter to compute the current address. Networking services for RTC synchronization, like NTP (Network Time Protocol) [1], are quite common. Modern NTP implementations use authenticated servers, which append a confirmation magic number to prove the authenticity of each synchronization packet. The use of authenticated NTP servers is warmly suggested for OTIP. Although misaligned clocks would not allow the discovery of the current IP addresses for servers, attackers could tweak the clients' perception of the current time, thus preventing the legitimate users from accessing their services. In other words, an attack on NTP would cause a DoS (Denial of Service) for OTIP.

NTP, or other RTC synchronization protocols, is able to reduce the error between the time read at different hosts below a certain predefined level. These protocols, in fact, periodically check the time of the current host against the time provided by reliable synchronization servers on the network. The difference between the local time and the time retrieved from the network corrected using an estimation of the communication delay, is used to put in place some modifying actions on the local RTC current value (e.g., by tuning the frequency of the local clock).

Even when the value of the current time can be retrieved from a very precise service, all the actions do not take place on clients and servers simultaneously, due to the transmission delays of the network. For example, a client's request takes time to travel across the network. Thus, the time read by the client when the request is issued will never be the same of the

time at which the server receives and processes that request.

In order to deal with all these errors and delays OTIP cannot manage the address change as an instantaneous action. For a specific time interval both addresses (the old one and the new one) should be valid. This time interval should be carefully chosen to solve two possible problems.

- A request from a client whose RTC is running slightly fast (while remaining within the admitted tolerance) may arrive in advance with respect to the time due for the address change on the server.
- A request from a client whose RTC is precise or slightly late can arrive at the server after the time due for address change, also because of the delay introduced by the networking communication.

In order for a server to tolerate clock desynchronization and network delays, the new address should be activated in advance for at least a time equal to the maximum difference between the RTC readings, and the old address should be kept valid for a time at least equal to the maximum transmission delay in the network plus the maximum difference of the RTC.

Formally, calling Δ_t the maximum desyncronization guaranteed by the clients and the server, i.e., if s is the server and c a generic client:

$$\forall c : |t_c - t_s| < \Delta_t \tag{1}$$

and when the maximum network delay is d_{net}, the period of the OTIP address change is T, then the n-th server address must be valid and available for clients in the time interval:

$$[t_0 + (n-1)T - \Delta_t, t_0 + nT + \Delta_t + d_{net}] \tag{2}$$

III. Related Work

A time based One Time Password (OTP) [2] is a password which is valid for a limited time. This security feature is commonly used to protect transactions on the network. Many Internet based banking systems, for instance, provide each user with their own small piece of hardware called *security token*. This device, usually similar to a key-holder, shows on its display a password generated on the basis of a secret seed and the current time provided by a reasonably precise clock which is a hardware component of the security token itself. OTIP applies the OTP concept to IP addresses instead of passwords.

IETF (Internet Engineering Task Force) have introduced by RFC3041 [3] and then by RFC4941 [4] the idea of dynamically changing IP addresses. The focus of these standards (and of other proposals like [5]) is the privacy of the client. Autoconfiguration methods, in fact, can reveal the MAC (medium access control) address of each host connected to the Internet because that address is copied in some bytes of the IPv6 address, allowing attackers to trace the position of a specific computer on the network. OTIP and these privacy extensions have different purposes: the former applies the dynamic change of IP addresses to protect server from attacks, the latters changes the IP addresses of the client to preserve the user's privacy.

OTIP is a *killer application* of the Internet of Threads (IoTh). As described in [6], IoTh opens a wide range of new applications by allowing each process to be a node of the Internet. Each process can have its own IP addresses, routing definitions etc.

Using IoTh each OTIP server can define its own OTIP policy, address change period, address computation algorithm, overlapping time interval for address validity etc.

The proof-of-concept implementation provided in section V, is based on LWIPv6 [7], View-OS [8] and msocket [9].

OTIP could be implemented without IoTh using a daemon to add and delete IP addresses of a network controller or a virtual interface of a container [10]. In both cases this daemon would be granted network administration capabilities (`CAP_NET_ADMIN` in Posix.1 [11] terminology). In this scenario the selection of the right address to use should be done by the `bind` system call. Other processes running on the same host may erroneously use the same address designed just for a specific server (e.g., by using `in6addr_any`, as many daemons do). Apart from the software architectural complexity of having a daemon as an executor of the OTIP address change requests, the effects of an attack would not be confined to a single service but could scale up to the container or to the whole networking support of the hosting system. IoTh implementation is clearer, simpler and safer. Finally, can be regarded as a special case of a hash-based address as defined in [12].

IV. Address Collision

It is clearly possible, although improbable, that two OTIP servers running on the same data-link network (real or virtual Local Area Network, LAN) temporarily get the same address. If we regard the addresses as if they were randomly chosen, this problem can be regarded as an application of the Birthday Problem (also known as the Birthday Paradox, as explained in [12]).

The probability of m nodes choosing the same address using a host suffix of h bits is:

$$Pr[(h,m)] = 1 - \frac{2^h! \binom{m}{2^h}}{m^{2^h}} \tag{3}$$

which can be approximated when $m << 2^h$:

$$Pr[(h,m)] \approx 1 - e^{-\frac{m^2}{2^{h+1}}} \tag{4}$$

Figure 1 shows the probability of address collision using a 64 bit network prefix and a 64 bit host address, which is the most common scenario in current IPv6 implementations.

The probability in a network connecting one thousand servers has the order of magnitude 10^{-14}, and even connecting one million servers the collision probability is less than 10^{-7}.

The effect of a collision is a temporary unreachability of the servers lasting for an OTIP address change period (64 seconds in the proof-of-concept implementation). Clearly this limitation should be taken into account for applications which require extremely stringent constraints in service continuity,

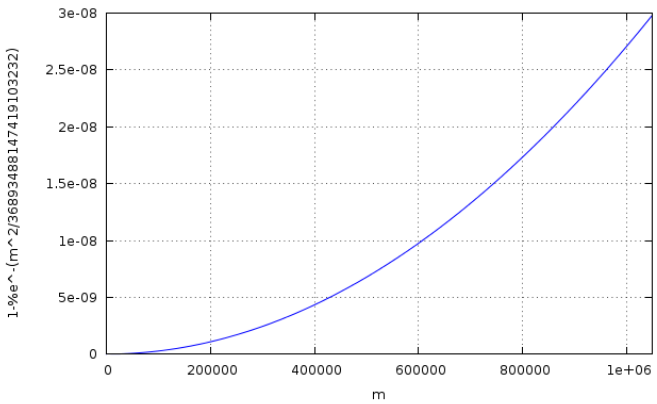

Fig. 1. Probability of address collision in a :64 IPv6 network [12]

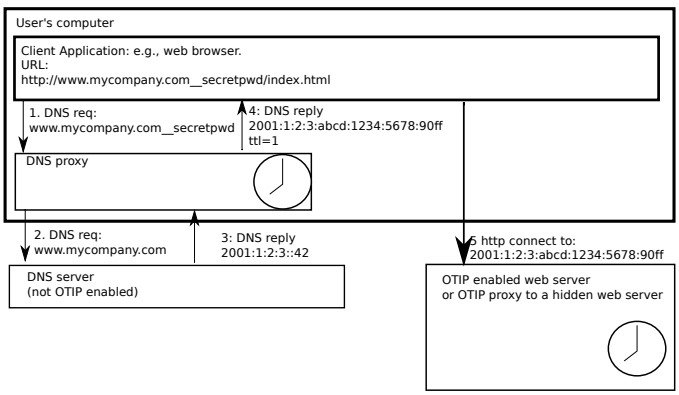

Fig. 2. Structure of the proof-of-concept implementation of OTIP for TCP communications

but the eventuality of service fault for address collision is able to satisfy the requirements of a very wide range of network services.

V. Implementation

This section describes a proof-of-concept implementation of OTIP for TCP (transmission control protocol) servers and some ideas for alternative implementations also for connection-less services based on UDP (user datagram protocol).

Figure 2 provides a schematic view of the test implementation. In order to support unmodified client programs the prototype uses an OTIP enabled Domain Name Server (DNS) proxy which is installed on the user's computer where the client program is running. This DNS proxy transparently forwards all the requests to the real DNS server except those matching the OTIP syntax. For the scope of this prototype the syntax is:

```
FQDN__password
```

So, if in a configuration file of the client program or, for instance, in the host part of a URL of a browser a specification like the following:

```
www.mycompany.com__secretpwd
```

appears, it will be resolved by the DNS proxy using the base address of `www.mycompany.com` retrieved by a query to the local DNS server (or possibly from the DNS server of `mycompany.com`). The host part of the resulting IPv6 address will be computed by the proxy using the base address, the password (`secretpwd` in this example) and the current time. Calling t the current system time in seconds (as returned by the `time` POSIX system call), the method implemented in this proof-of-concept computes a 128bit MD5 hash of the string composed by $t \gg 6$ (right shifted 6 bit positions, i.e., divided by 64), a space and the password. The statement used to create the input string for the hash function is the following:

```
len=asprintf(&s,"%d %s",time(&now)>>6,pwd);
```

The result of an exclusive-or (XOR) operation between three operands:

- the 64 most significant bits of the MD5 hash value,
- the 64 least significant bits of the same value,
- and the host part of the address returned by the real DNS,

becomes the host part of the current OTIP address. It is worth noting that the description of the algorithm used to compute the OTIP address in this implementation has been given here for the sake of presentation completeness. Any choice of hash function involving the time and the FQDN or IP address returned by the real DNS would fit, provided the same algorithm is consistently applied both by the server and the client.

The proxy sets the TTL (time-to-live) to one second in its reply to force the client to invalidate the resolution cache in a short time and to repeat the query for any further connection needed to the same server. This is necessary as the address may have changed in the meanwhile following the OTIP specifications. The password is never sent along the network as the proxy is running in the same host of the client process, thus it cannot be captured by an attacker possibly tracing the network traffic.

On the server side either an OTIP specific server program or a proxy interfacing an existing server daemon, provide the connectivity for OTIP clients. The Berkeley sockets and msocket API (application programming interface) use the `accept` system call to manage each TCP incoming connection `accept` takes as its first argument a socket descriptor used for listening to the arrival of new connections (listening socket) and returns a new socket descriptor connected to the remote client (connected socket) when a new connection arrives. When the address validity expires, OTIP closes the listening socket so that no new connections can take place using the old address, and it opens a new listening socket using the new address. Connected sockets continue to use the address which was the current one at their connection time. An address is completely dismissed when the last connection using that address gets closed.

It is easier to code an OTIP enabled server or an OTIP proxy using msocket API, as it is possible to start and close

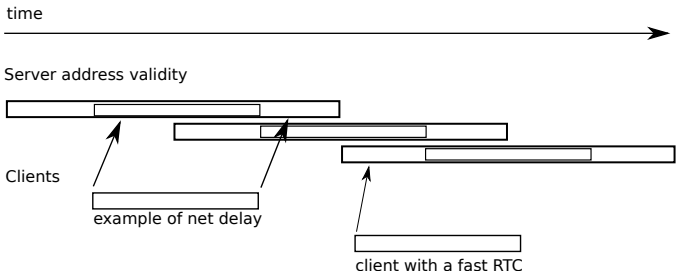

Fig. 3. Server address intervals of validity (as perceived from the server and by the clients)

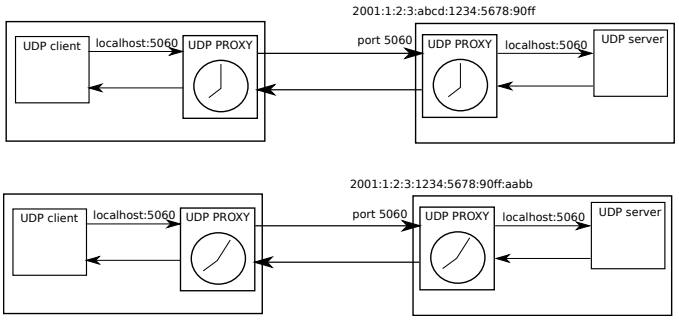

Fig. 4. A proposal to support OTIP for UDP services using existing clients and servers.

an entire TCP-IP stack each time a new address is needed or an old address expires. In the proof-of-concept implementation each address has a validity period of 64 seconds. On the server side each address is valid for 128 seconds. It is activated 32 seconds before the beginning of its validity period defined for the client and it expires 32 seconds after the end of the period as shown in Figure 3.

The overlapping between the validity periods of successive addresses has been set to 32 seconds, which should normally be a value far beyond the sum of the maximum difference between the clock readings plus the maximum transmission delay using NTP and a non-overloaded network.

In theory, this choice may reduce the time needed for a brute force attack by a factor less than four (two addresses always active for double the time). De-facto, 2^{62} attempts (which is $\sim 4\ 10^{18}$) to find an address which needs to be exploited in less than two minutes is a specification able to satisfy the security requirements of a wide range of applications.

The method described above for TCP cannot be directly applied to UDP. UDP is connection-less so it is not possible to keep the old address working only for the data exchanged along the existing connections. It is possible to write OTIP aware UDP clients which take care of the OTIP address validity periods and compute the new server address as needed. As a backwards compatibility feature, it is possible to implement OTIP UDP proxies which set up an OTIP UDP tunnel between the client and the server. Each client uses a port on the local host to exchange networking packets to the server. The local port is managed by the local proxy, which forwards each packet to the OTIP dynamically changing address of the proxy on the server side. On the server side, the OTIP UDP proxy forwards the packets to the OTIP unaware server using a local port (see Figure 4). Proxies are able to support several UDP clients at the same time. To do so, a new port is used by the client side proxies for each local client and by the server side proxies for each remote client. In this way the return packet can be properly rerouted.

VI. Limits, Future Developments and Conclusions

OTIP reduces the vulnerability of private services provided through the Internet. It makes it hard for attackers to discover the current address used by servers to communicate with their legitimate users. This method should be applied, together with other already known precautions, to protect communications along the Internet. A non-exhaustive list of such defenses, which should be put in place, includes the encryption of the communication payloads and the measures to avoid TCP spoofing, as defined in RFC 4953 [13]. In fact, an attack on the TCP sequence numbers could permit an attacker to *hijack* an existing TCP connection, avoiding, in this way, the OTIP protection.

Another scenario in which the protection of OTIP can be disrupted happens when even one user's host running the client program and the DNS proxy is compromised. An intruder who installed a root-kit on a host could read the OTIP passwords and compute the current addresses of the servers at a later time.

The implementation section above has shown how existing networking clients and servers can already use OTIP. Some programs may not be able to take advantage of this support. During the studies and tests we have seen two cases of programs which need some changes to use OTIP. In one case a TCP client did not properly managed the value of TTL returned by the DNS query and used a cached address far beyond its expiration. Unfortunately, such behavior is not very rare, as the server address is perceived by programmers as a constant. A second case happens when it was not possible to configure the client to change the address of the server for UDP communications, in order to use the UDP proxy. Sometimes the change of the server address is not a configurable entity in itself as it is the same address used for other services, or the UDP server address was communicated to the client as an element of the protocol.

As far as the address collision problem is concerned, we have shown its very limited and temporary impact. This drawback, however, cannot be easily eliminated by adding a duplicate address detection check, as proposed for the IPV6 privacy extensions [4], as servers and clients compute the current addresses independently. The clients cannot locally detect duplicate addresses on the servers' networks, and any sub protocol designed to force the clients to change the address sequence would weaken the methods, as it could also be used by attackers.

Although OTIP can be applied as described in section V of this paper, the syntax and implementation of OTIP should be standardized to provide a general purpose support to this new

feature. OTIP does not need to change any existing protocols, and it is able to support many existing client programs. Some standardization effort is needed, for example, to share the same OTIP DNS proxy for all the OTIP servers, as different service providers could use a different syntax or a different function to compute the current address.

It is worth noting that OTIP does not exclude hash based address definition as introduced in [12]. System administrators use hash based addresses to configure their hosts and DNS servers in an easier and less error-prone mode. Hash addresses can be used as base addresses for OTIP, combining a simple deployment of the network at the servers' side and the safety of the dynamic evolution of addresses as provided by OTIP.

The source code to test the experiments presented in this paper can be downloaded from `svn://svn.code.sf.net/p/view-os/code/branches/otiptest` and has been released under the GNU General Public License (GPL) v. 2 or newer. The programs are intended as just a proof-of-concept to show the effectiveness of the ideas introduced here.

REFERENCES

[1] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification," RFC 5905 (Proposed Standard), Internet Engineering Task Force, Jun. 2010.

[2] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, "TOTP: Time-Based One-Time Password Algorithm," RFC 6238 (Informational), Internet Engineering Task Force, May 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6238.txt (Retrieved: June 15, 2013)

[3] T. Narten and R. Draves, "Rfc 3041: Privacy extensions for stateless address autoconfiguration in ipv6," IETF, Tech. Rep., 2001.

[4] T. Narten, R. Draves, and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 4941 (Draft Standard), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4941.txt (Retrieved: June 15, 2013)

[5] M. Tortonesi and R. Davoli, "User untraceability in next-generation internet: a proposal," in *Proceeding of Communication and Computer Networks 2002 (CCN 2002)*, IASTED, Ed., November 2002, pp. 177 – 182.

[6] R. Davoli, "Internet of threads," in *Proc. of the The Eighth International Conference on Internet and Web Applications and Services, ICIW 2013.*, 2013, pp. 100–105.

[7] ——, "LWIPV6," http://wiki.virtualsquare.org/wiki/index.php/LWIPV6 (Retrieved: June 15, 2013), 2007.

[8] L. Gardenghi, M. Goldweber, and R. Davoli, "View-os: A new unifying approach against the global view assumption," in *Proceedings of the 8th international conference on Computational Science, Part I*, ser. ICCS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 287–296.

[9] R. Davoli and M. Goldweber, "msocket: multiple stack support for the berkeley socket api," in *SAC '12: Proceedings of the 27th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2012, pp. 588–593.

[10] LXC team, "lxc linux containers," http://lxc.sourceforge.net/ (Retrieved: June 15, 2013).

[11] POSIX.1-2008, "The Open Group Base Specifications," Also published as IEEE Std 1003.1-2008, San Francisco, CA, Jul. 2008.

[12] R. Davoli, "Ipv6 hash-based addresses for simple network deployment," in *Proc. of the The Fifth International Conference on Advances in Future Internet, AFIN 2013, To appear*, 2013.

[13] J. Touch, "Defending TCP Against Spoofing Attacks," RFC 4953 (Informational), Internet Engineering Task Force, Jul. 2007.