The ISP Column

An occasional column on things Internet

September 2007

Geoff Huston

Trust

Trust and networking go hand in hand, and I'm pleased to see that the topic of trust has been raised by the Internet Society recently.

> Internet Society Board of Trustees
>
> Call for Participation: Trust and the Future of the Internet
>
> The Internet Society (ISOC) Board of Trustees is currently engaged in a discovery process to define a long term Major Strategic Initiative to ensure that the Internet of the future remains accessible to everyone. The Board believes that Trust is an essential component of all successful relationships and that an erosion of Trust: in individuals, networks, or computing platforms, will undermine the continued health and success of the Internet.
>
> The Board will meet in special session the first week October of 2007 for intensive study focused on the subject of trust within the context of network enabled relationships. As part of this process, the Board is hereby issuing a call for subject experts who can participate in the two day discussion. Topics of interest include: the changing nature of trust, security, privacy, control and protection of personal data, methods for establishing authenticity and providing assurance, management of threats, and dealing with unwanted traffic.
>
> [*http://www.isoc.org/isoc/general/trustees/headlines/20070809.shtml*]

In this column I'd like to informally respond to this call for participation and revise some earlier thoughts I had on trust to see if we've made any significant progress on the issue of trust in the Internet in the past four years.

These days the Internet assumes a role of underpinning all kinds of commercial, governmental and personal activities, and its important to all that it operates as reliable and trustable service. The question being posed here is can the services you encounter on the Internet be trusted? Are the service edifices we've constructed in the Internet anchored upon solid foundations?

How robust is the Internet? How resilient is the Internet to attempts to disrupt it? Interestingly enough, the early work in packet switched networks was focused on exploring the notion that it could be possible to construct a system that was more reliable than its components.

> The Internet is undoubtedly the unintended outcome of the initial research objectives articulated by the Advanced Research Projects Agency (ARPA) of the United States Department of Defense (DoD) in the late 1960s, but the antecedents of this effort go back a few years earlier in the U.S. research community. The so-called think-tank of the cold war, the RAND Corporation, was an early vehicle for the concept of computer networking. There, Paul Baran, whom many consider to be the father of computer networking, presented his ideas on the subject in a seminal work published by RAND in 1964, "On Distributed Communications"
>
> "...one day we will require more capacity for data transmission than needed for analog voice transmission. ... it would appear prudent to broaden our planning consideration to include new concepts for future data network directions. Otherwise, we may stumble into being boxed in with the

uncomfortable restraints of communications links and switches originally designed for high quality analog transmission. New digital computer techniques using redundancy make cheap unreliable links potentially usable. A new switched network compatible with these links appears appropriate to meet the upcoming demand for digital service. This network is best designed for data transmission and for survivability at the outset."

[*"On Distributed Communications: I - Introduction to Distributed Communications Networks," RM-3420-PR, August 1964*]

The entire set of Baran's classic RAND papers on packet-switching from the early 1960s are at http://www.rand.org/publications/RM/baran.list.html.

However it is perhaps only in theory that there is no difference between theory and practice. The practical experience of the Internet reveals a disturbingly rich history of disruptive attack. Perhaps the most noteable early recorded attack was the "Internet Worm" of the late 1980's which exploited a bug in a number of commonly deployed utility programs on Unix hosts.

This worm had a number of attack modes, including exploiting weaknesses in the sendmail daemon as well as the "finger" daemon. The finger daemon did not protect itself against buffer overflow. Since then documentation of the "gets" library routine includes dire warnings that this is a dangerous function and should not be used

"This is a _dangerous_ function, as it has no way of checking the amount of space available in BUF. One of the attacks used by the Internet Worm of 1988 used this to overrun a buffer allocated on the stack of the finger daemon and overwrite the return address, causing the daemon to execute code downloaded into it over the connection." [gets manual page, FreeBSD v4.7 documentation]

A copy of the paper on the "A Tour of the Internet Worm" by Donn Seeley can be found at http://world.std.com/~franl/worm.html

Since then we've seen a steady stream of attacks on the Domain Name Service, attacks on hosts using TCP SYN flooding, viruses, distributed denial of service attacks, attacks on the routing system, Code Red and its variants, SNMP, and the Slammer attack on SQL servers, to point to but a few. Why have we been unable to eliminate such vulnerabilities?

All networks have an inherent trust model. When two parties cannot directly interact with each other to confirm their respective identities, roles and authorities, then they are forced to place some element of trust in the intervening network. In some cases the trust model relies on the ability of the network provider to operate the network correctly. For example, when I enter a string of digits into the numeric pad of my telephone I'm trusting the network to create a connection to a particular handset somewhere else in the network. In other cases the trust model relies on each user acting appropriately, and the network provider is in no position to enforce such user behaviour.

At a basic level the Internet is a simple datagram delivery network. The network itself does not define any service other than packet delivery. Surrounding the network is a collection of end-systems. It is these systems that define the service model of the Internet, and define how the network resources are to be shared across competing demands. Some of these systems are operated by the same entity that manages the network, but the overwhelming majority of such systems are operated by other users of the network. In the Internet's trust model every user trusts, to some extent, the benign intent of every other Internet user. In a closed homogenous small community this trust may be well-placed. In the broader context of a public utility system with hundreds of millions of users, this is stretching the friendship model a little bit beyond its natural limits.

To illustrate this distributed trust model, lets look at the action of download a web page using a browser.

In this example I've collected the IP packets from a local host as it retrieves the web page http://www.potaroo.net.

The local host is a Windows XP operating system, with IPv6 enabled.

Starting a Session: DNS Trust

Once the URL is entered into the browser, the browser's first task is to translate the domain part of the URL into an IP address. The browser passes the domain name to the local DNS service routines and awaits a response. The local DNS service routine has been configured with the IP address of a remote resolver. The IP address of this DNS resolver has been provided automatically as part of the initial connection setup using the DHCP protocol, or it's been manually configured into the local system. The local host asks its DNS resolver to translate the name to an IP address.

The Windows XP TCP driver appears to want to establish a IPv6 connection with the web server first, if it can be achieved, and if not, then drop back to establishing an IPv4 connection.

So, using an IPv4 UDP transport, the client queries the local server for a V6 address for the domain name, "www.potaroo.net"

The first packet was sent from the local host "gih" and the packet was directed to a local DNS name resolver, "named", using UDP port 53 on that system. The payload of the UDP packet is a DNS query, with a request for an AAAA DNS Resource Record (IPv6 address) for the name "www.potaroo.net" :

```
  IPv4 UDP gih.1030 > named.53: 16+ q: AAAA? www.potaroo.net.
```

The response is that while the domain exists, there is no V6 address associated with the name:

```
  IPv4 UDP named.53 > gih.1030:  16 q: AAAA? www.potaroo.net.
    0/1/0/0 ns: sec1.apnic.net SOA potaroo.net. gih.potaroo.net.
    2007082201 10800 3600 3600000 6400
```

If the IPv6 query had succeeded in returning an IPv6 address, the Windows XP TCP protocol stack would attempt to open the TCP session by sending a TCP SYN packet with a 3 second timeout. If there is no response in the 3 seconds it sends another packet with a 6 second timeout, and if still no answer it will send a third SYN packet with a 12 second timeout. Only then will it drop back to attempting an IPv4 DNS query.

In our case the IPv6 DNS query did not return an IPv6 address, so its time to try IPv4, and ask for an IPv4 address (or A Resource Record) from the DNS:

```
  IPv4 UDP gih.1030 > named.53: 17+ q: A? www.potaroo.net.
```

This time the DNS returned the address record of 203.119.0.116:

```
  IPv4 UDP named.53 > gih.1030:  17 q: A? www.potaroo.net.
    1/1/0/0 www.potaroo.net A 203.119.0.116
```

So the answer is that the domain name “www.potaroo.net” maps to the IP address 203.119.0.116.

This is a very simple protocol – the local system poses a simple query to a name resolver in a single packet, and it gets back an answer from that name resolver in a single packet. There are no other credentials associated with the answer, so there is nothing for the local system to check to validate the answer. So we simply have to trust the accuracy of this answer.

Is this trust well-placed?

The DNS is a highly distributed database, where various components of the database are operated by a diverse collection of operators. It would be comforting to believe that the DNS provides accurate answers to queries all of the time. This is not the case.

It would possibly be an acceptable compromise to believe that all incorrect answers are the result of temporary faults or inadvertent operator errors. Unfortunately even this is not the case all of the time.

The DNS is a target of various forms of attack, and in some cases such attacks are successful. In such cases the DNS provides incorrect answers, directing the user to a site that may then attempt to compromise any ensuing transaction.

- It could be that the name resolver has been compromised, and is delivering incorrect answers.
- It could be that an eavesdropper on the path between my system and my name resolver has seen the query and injected a fake response.
- It could be that the name resolver is using a forwarder, and this forwarder has been compromised.
- It could be that the primary zone for potaroo.net has been updated and the secondary servers for this zone have not yet refreshed their local copy and the information is out of date.
- It could be that some part of the routing system has been compromised and DNS traffic addressed towards the DNS servers for potaroo.net is being redirected towards a different server that is providing incorrect answers, while masquerading as the authoritative server for this domain.
- If could be that this redirection has happened for the .net servers, or even the root DNS servers.
- It may be that the DNS servers are using some form of load distribution technique to spread the query load over a number of servers, and the distribution mechanism has been compromised.

And no doubt there are many more uncomfortable vulnerabilities in the operation of the DNS.

The inevitable conclusion is that DNS is not a secure protocol and the underlying data is not well secured. There is no means of ensuring that the data one gets back is authentic. As noted in a threat analysis of the DNS there are a number of attack modes for the DNS which are undetectable by the victim.

The threat modes listed in a study of DNS threats, in RFC3833, included packet interception, where the simple unencrypted two packet exchange protocol of the DNS makes the system vulnerable to various forms of man-in-the-middle attacks, including eavesdropping with false responses injected back into the network in advance of the actual responses. It is also possible to apply various name games to DNS responses. The DNS relies on extensive use of caching of answers to improve its performance and reduce load on the DNS servers. One form of attack is to feed false data into the cache, which in turn triggers the local system to then query the fake DNS servers. The threat is based on passing back to the query point a DNS response that includes one or more DNS resource records in the Additional section of the DNS response.

As RFC3833 points out: "The common thread in all of the name chaining attacks is that response messages allow the attacker to introduce arbitrary DNS names of the attacker's choosing and provide further information that

> the attacker claims is associated with those names; unless the victim has better knowledge of the data associated with those names, the victim is going to have a hard time defending against this class of attacks.
>
> This class of attack is particularly insidious given that it's quite easy for an attacker to provoke a victim into querying for a particular name of the attacker's choosing"
>
> [*RFC3833 "DNS Threat Analysis", Atkins and Austein, August 2004*]

Setting up the Transport: TCP Trust

Once the DNS returns an IP address, and assuming that I'm willing to trust the DNS answer, the next step is to open an HTTP session. HTTP is layered on top of TCP, so the first step is to send a TCP SYN packet to the IP address to start the connection.

Send a TCP SYN packet to port 80 of the server

```
  IPv4 TCP gih.1044 > www.potaroo.net.80: SYN Seq=0 Ack=0  Win=32767 Len=0 MSS=1260
```

The server should accept the connect request by responding with an ACK of our SYN packet, plus a SYN packet of its own, which it does.

```
  IPv4 TCP www.potaroo.net.80 > gih.1044: SYN,ACK Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
```

We complete the handshake by sending an ACK of the server's syn packet.

```
  IPv4 TCP gih.1044 > www.potaroo.net.80: ACK Seq=1 Ack=1 Win=34020 Len=0
```

The TCP session is now connected.

We've set up a transport session with the web server www.potaroo.net and now we're ready to undertake a fetch.

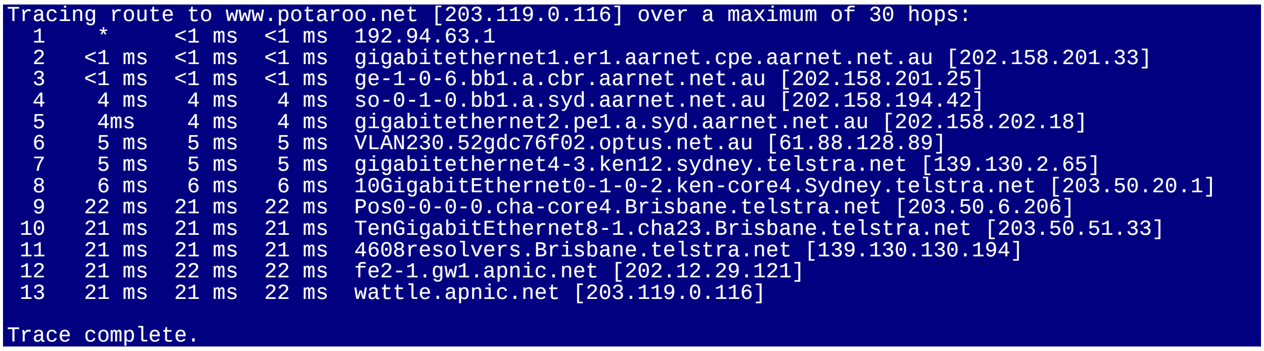
But are we sure that's what's happening? Are we talking to the 'right' web server? Again trust comes into play.

Here we are trusting the integrity of the Internet's routing system. The routing system is vulnerable to incorrect routing information being injected into the routing system, and as a consequence packets may be misdirected. While the majority of such incidents are attributable to operator error, there remains the potential that the routing system is vulnerable to deliberate attack. While we are sending packets to the IP address 203.119.0.116, if the routing system has been attacked there may be a false advertisement for this IP address, and my packets may be sent to a system that is masquerading as the web server, but is actually a fake. And I can't tell.

How easy is it to inject false information into the routing system and not be detected? There is no way to validate routing information, so each router in effect is forced to trust its peers that the information being provided is correct. Routing protocols typically contain no specific mechanisms to prevent the unauthorized modification of the information by a forwarding agent, allowing routing information to be modified, deleted or false information to be inserted without the knowledge of the originator of the routing information or any of the recipients. There is also the potential for injection of false information, interception, hijacking, and denial of service to disrupt the operation of the routing system.

> RFC4593 enumerates a relatively expansive set of set of threats to routing protocols and the consequences of these threats. The document graphically illustrates the extent to which routing exhibits a “weakest link” vulnerability, where the entire system is vulnerable to the risks that are exposed by the most vulnerable element.
>
> [*RFC4593, “Generic Threats to Routing Protocols”, Barbir et al, October 2006*]

Of course even if the routing system is operating correctly, the end user is also trusting the integrity of the various forwarding devices along the path to ensure that the packet is being delivered to the correct server in accordance with the local forwarding decision as determined by the routing system. Again this trust may be misplaced.



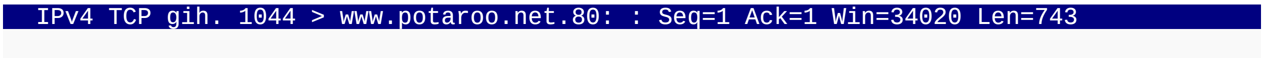
```
Tracing route to www.potaroo.net [203.119.0.116] over a maximum of 30 hops:
  1     *      <1 ms  <1 ms  192.94.63.1
  2   <1 ms  <1 ms  <1 ms  gigabitethernet1.er1.aarnet.cpe.aarnet.net.au [202.158.201.33]
  3   <1 ms  <1 ms  <1 ms  ge-1-0-6.bb1.a.cbr.aarnet.net.au [202.158.201.25]
  4    4 ms   4 ms   4 ms  so-0-1-0.bb1.a.syd.aarnet.net.au [202.158.194.42]
  5    4ms    4 ms   4 ms  gigabitethernet2.pe1.a.syd.aarnet.net.au [202.158.202.18]
  6    5 ms   5 ms   5 ms  VLAN230.52gdc76f02.optus.net.au [61.88.128.89]
  7    5 ms   5 ms   5 ms  gigabitethernet4-3.ken12.sydney.telstra.net [139.130.2.65]
  8    6 ms   6 ms   6 ms  10GigabitEthernet0-1-0-2.ken-core4.Sydney.telstra.net [203.50.20.1]
  9   22 ms  21 ms  22 ms  Pos0-0-0-0.cha-core4.Brisbane.telstra.net [203.50.6.206]
 10   21 ms  21 ms  21 ms  TenGigabitEthernet8-1.cha23.Brisbane.telstra.net [203.50.51.33]
 11   21 ms  21 ms  21 ms  4608resolvers.Brisbane.telstra.net [139.130.130.194]
 12   21 ms  22 ms  22 ms  fe2-1.gw1.apnic.net [202.12.29.121]
 13   21 ms  21 ms  22 ms  wattle.apnic.net [203.119.0.116]

Trace complete.
```

In this case the network path is forwarded through 12 switching units. There is the assumption that the packets are not being inspected while in flight, with the potential for fake responses to be generated before the remote system can respond. Again, it is not clear that such trust in the confidentiality of packet transit in the network is well placed all of the time.

Of course it could be that my session is being deliberately hijacked without my knowledge. It could be that the packets are being passed to a so-called *transparent proxy web cache*. In such a case the packet exchange will look precisely the same, and the received packets will use the source address of the target server. Even a traceroute to the server will show a viable network path to the intended destination, but the TCP session has not been established with the host at the IP address in question. Instead, when a transparent proxy web cache has been deployed, the forwarding system redirects the clients TCP port 80 packets into the cache server and the cache server is masquerading as the intended server. In that case I’m not really talking to the web server at *www.potaroo.net* at all, but to some other system that I cannot identify. I really cannot tell that my session has been hijacked in this manner unless I look in the log files of the web server to find a record of my download. Proxy web caches were intended to improve the performance of the web. But at the same time they introduce some troubling questions of trust. Has the cache server been compromised? Is the cached copy of the web page an accurate mirror of the current contents of the original or has it been altered in some way? This initial TCP packet exchange provides no reassurance that the connection being set up is the intended connection with the server that is hosting the desired web page. Packet hijacking has happened in such a case and while its intended to be a benign hijack, all I can do is trust in the integrity of the proxy setup. Again, it’s an uncomfortable stretch of the trust model.

Delivering the data: HTTP Trust

When the TCP session is opened, the user's HTTP session then requests the web object from the server.



```
  IPv4 TCP gih. 1044 > www.potaroo.net.80: : Seq=1 Ack=1 Win=34020 Len=743
```

This contains a payload, which is the initial HTTP request:

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
        application/vnd.ms-excel, application/vnd.ms-powerpoint,
        application/msword, application/x-shockwave-flash, */*
Accept-Language: en-au,en-us;q=0.5
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
            .NET CLR 1.1.4322; Infopath.1; .NET CLR 2.0.50727)
Host:.www.potaroo.net
Connection: Keep-Alive
```

HTTP is very chatty isn't it?

Identifying the CPU type of the local host and the user agent is perhaps being a little too chatty. If the version of the user agent on this CPU architecture has a know exploit then telling every server that you are vulnerable is perhaps extending trust too far.

It also appears to be letting the server know what applications can be fired up within a response. Again not exactly reassuring information to be passing around to strangers if there is a means of using web response that can trigger a vulnerability within one of these applications.

Of course it could include even more information. When a link is being followed then the request also includes a Referer: field, referencing the URL of the page containing the link. The Referer header allows reading patterns to be studied and reverse links drawn. Although it can be very useful, its power can be abused if user details are not separated from the information contained in the Referer. Even when the personal information has been removed, the Referer header might indicate a private document's URI whose publication would be inappropriate.

> To quote from the Security Considerations section of the HTTP 1.1 protocol specification:
>
> “HTTP clients are often privy to large amounts of personal information (e.g. the user's name, location, mail address, passwords, encryption keys, etc.), and SHOULD be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources. We very strongly recommend that a convenient interface be provided for the user to control dissemination of such information, and that designers and implementors be particularly careful in this area. History shows that errors in this area often create serious security and/or privacy problems and generate highly adverse publicity for the implementor's company. “
>
> [*RFC2616, “Hypertext Transfer Protocol – HTTP/1.1”, Fielding et al, June 1999*]

The server acknowledges the input and then commences download of the web page data using HTTP over TCP.

Is the object delivered by the server that original object that was placed there by the content originator?

Servers are vulnerable to attack, and defacing a web site by substituting altered content in the place of the original is not uncommon. While some forms of content substitution are readily apparent to the user, other forms of substitution may be far more subtle in their intent, attempting to convince the user that the content is genuine and that the web transaction is trustworthy. Other forms of attack to the web transaction are more difficult to trace. If the user's request is passed through a web cache, then an attack on the cache can result in the delivery of substituted content to the user while the content originator's site has maintained its integrity.

So when you enter a URL into a browser, and get back a page there's a lot of trust is happening.

Probably too much trust.

If that's the case, then how can we improve the situation?

Replacing Trust with …?

The basic toolset when looking at how to replace trust is all about authentication and validation, together with the ability to be able to conduct a private data exchange even across a public medium. Paradoxically, many of the basic tools are already available.

But being available and being used are different concepts.

The first set of tools is all about using cryptography to support the concept of a private data exchange within an untrusted medium. There are effective cryptographic tools in widespread use that can both conceal the contents of the conversation from third parties and clearly flag attempts to alter or disrupt the conversation. Whether its the Data Encryption Standard (DES), Triple DES, the International Data Encryption Algorithm (IDEA), the Advanced Encryption Standard (AES), Blowfish, Twofish or any other block cipher, it is possible to encrypt a data exchange between two parties to a level where there is reasonable confidence that the data will remain private to the parties in the conversation.

Of course, as with many other secure systems, its not the encryption algorithm that is the vulnerable point, but the initial key exchange that sets up the communication using a shared secret, namely the encryption key. If this key exchange is compromised, then so is the integrity of the communication.

The common solution to this is the concept of private and public key pairs. While the mathematics behind this is well beyond the scope of this article, the essential characteristics of a public / private key pair is that anything encoded using the private key can only be decoded using the corresponding public key, and anything encoded using the public key and only be decoded using the private key. And, most importantly, knowledge of the public key will not allow you to compute the value of the private key.

Now we have the elements of a more robust environment. If you know my public key you can encode some data using this public key and pass it to me. Only my private key can decode it, and, as long as I keep this private key a secret then only I can read your original data. And if you then encode this data using your private key, then I will need to decode the data first with your public key and then with my private key. At this point we have a mechanism to undertake a relatively secure form of key exchange, where each party can identify the other, and exchange messages that only the other party can read.

This approach hinges on being able to associate a public key with the remote party in a trustable manner. And here's where a trustable public key infrastructure comes into play.

Widespread use of a trustable public key infrastructure (PKI) would assist in allowing users to validate that the content has been generated by the author. In a model of complementary public and private key pairs a web object can be signed using the object originator's private key. If the object's signature can be validated against the originator's public key then there is a strong indication that the content is genuine and has not been altered in any way.

Imagine for a second if a public key infrastructure was in widespread use and we only accepted email that was digitally signed using keys that we trust. Imagine also that web objects were also signed in a similar fashion. In such a world how would a spammer hide their true identity? If spammers cannot hide behind anonymity, then would spam be as prevalent in such a network? Would various ecommerce applications enjoy greater acceptance if users were confidant that there was both privacy and authenticity associated with such transactions? What would be the point of subverting a web site or its content if the action were immediately visible as an unauthorized alteration? Not only would such a network mitigate some of the problems we see today, but its likely that we would find new services appearing that rely on a useful end-to-end trust model where each end can validate the identity of the other party and also validate the authenticity and privacy of the exchanged data.

We can go beyond the application level in this, and look at how we can improve upon the trust model in the network itself and its associated service layer.

A comprehensively deployed DNSSEC model within the DNS, from the root all the way down to each and every domain name could ensure that any DNS responses can be validated, and the DNS could be transformed form one of the major points of vulnerability in the Internet today into one of its more secure and robust components. In such a complete DNSSEC world where DNSSEC were to be an integral part of the operation of the DNS, it would be a far more difficult proposition to attempt to fudge bad data into the DNS.

It would be useful to have each registry-allocated address block be associated with a public key infrastructure and for all routing updates to be appropriate signed in the context of this PKI. This would allow routing information to be validated which in turn would mitigate those forms of hostile attack that are based upon injection of false routing information into the network.

Its not that we can eliminate the need for some degree of trust from the Internet, nor that all potential security risks can be comprehensively addressed. But in using a set of basic security tools with our existing network applications we can make some significant improvements upon the current open trust model of the Internet. In so doing we can make significant improvements in the level of trust users can reasonably place on the integrity of the Internet as a useful communications medium. And surely that's a good thing.

But while this all sounds excellent in theory, in practice we have yet to see public key infrastructure approaches gain any traction in the Internet. Even the efforts we've made so far in this area are unsatisfying upon closer inspection. The trust certificates that are loaded into my browser in order to create that padlock icon when I visit a “secure” web site are all issued by entities I have no direct knowledge or, and whose business model when they issue certificates is completely unknown to me. I have no idea how the association between a Certificate Authority, a certificate holder and a particular domain name was created, and all I can do is trust that the “right” thing has happened, without even knowing what that “right” thing may have been in any case!

While its easy to bemoan the fact that Internet users are lazy, uninformed and just plain recalcitrant, and they don't place sufficient value in robust security measures, and are unwilling to use them, to test them, let alone pay a premium for them, I suspect that the failing in this approach to securing the network goes a little deeper than just blaming the users for being lazy and ill-informed.

Trust is undoubtedly the cheap default answer, and most of the time this trust appears to be well-placed. When we click on a link, enter a URL into a browser or any one of a hundred different application level actions that result in network transactions we all do each and every day, then the expected thing usually happens. So for most of us, most of the time, this level of trust just works. So why should you or I be expected to pay a premium for more complicated applications, more convoluted procedures and more restricted network environments to mitigate the relatively slight risk of hostile attack? Good security does not appear to present us with insurmountable technical challenges, but it surely has presented us with a close to intractable business challenge. Trust is just too cheap and, most of the time, trust is just good enough.

So is trust the universal answer?

The problem for me is that “trust” is not that much different from blind faith, and, in that light, “trust” is not a very satisfying answer. The difference between “fortuitous trust” and “misplaced trust” is often just a matter of pure luck, and that's just not good enough for a useful and valuable network infrastructure. “Trust” needs to be replaced with the capability for deterministic validation of actions and outcomes of network-based service transactions. In other words, what is needed is less trust and better security.

But if good security is ever going to be the generally adopted answer, then somehow we have to strive to replace today's trust with security measures that somehow manage to achieve that mythical triumvirate of networking: Good, Fast and Cheap!

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

About the Author

GEOFF HUSTON holds a B.Sc. and a M.Sc. from the Australian National University. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. He is author of a number of Internet-related books, and is currently the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He was a member of the Internet Architecture Board from 1999 until 2005, and served on the Board of the Internet Society from 1992 until 2001.

http://www.potaroo.net