;login:

THE MAGAZINE OF USENIX & SAGE

November 2000 • volume 25 • number 7

THEME ISSUE: SECURITY
edited by Rik Farrow

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

in this issue

by Rik Farrow

Theme Issue Editor

<rik@spirit.com>

For the ninth time, USENIX and its members created a security symposium. In the past, two years had to pass between conferences. Security was not such a big deal. But no longer. Chairs will always say that this conference is the best ever, and it seems this time that it was true. Wonderful papers, marvelous speakers.

Win Treese put together the Invited Talks track. If you ever want to do something that is nontechnical and really challenging, you should help put together an Invited track. Treese went so far as to add a very human element to the conference by bringing in Suelette Dreyfus to speak on Cryptography and Human Rights. Dave Dittrich scared us all by reminding us that Distributed Denial of Service attacks have not stopped – they just don't make the news these days. Duncan Campbell talked about Echelon for the conspiracy buffs in attendance (and there were quite a few), and Mudge explained how a tool written by the l0pht (now @stake) goes about detecting network interfaces listening promiscuously.

You can find all of the papers online, of course, but I wanted to mention a few. Publius is an interesting effort to permit anonymous and irrevocable publication of sensitive documents. The detecting-backdoors pair of papers (one of which won Best Student Paper) examine a way to detect backdoors and relays using network heuristics and headers only, so that it is not necessary to sniff the data portion of packets, which might be encrypted anyway. You should read the actual papers if you are interested, as well as the summaries in this issue, to get a good idea of the papers, Invited Talks, and a couple of BoFs.

I do not mean to slight any of the paper writers, as I found them excellent reading, especially while sitting in the Denver airport during hours of thunderstorms, waiting for United to allow their planes to approach the passenger tunnels. Isn't it amazing just how fragile our technology base is?

This issue of *;login:* contains feature articles that I solicited from the security community. I especially wanted a better understanding of DNSSec. It is one thing to read the RFCs and quite another to talk to someone, in this case Evi Nemeth, who has played with implementations, written a chapter in a book, and had BIND 9 implementors edit the article you will find here.

I asked David Brumley and Steve Romig if they would contribute again. David already had an idea in mind, the technique that he is using to help secure Linux systems at Stanford University (through the creation of a secured distribution so that people can at least start right). Steve has been working in computer investigations for years and will teach a tutorial at the USENIX conference in San Diego in early December. Steve writes about collating logs and understanding how they are used as evidence.

The Nessus team in France contributed an article about their vulnerability scanner. These guys have put together an open source tool with a large collection of vulnerabilities (no exploits, kiddies), and their tool ranked number one in Fyodor's survey of the best security tools. (ISS ranked seventh, not bad for one of three commercial products mentioned, but pretty poor when you consider that Nessus is free.) Fyodor's list, at <*http://www.insecure.org/tools.html*>, is a nice reference for security tools (50 of them, although the link for VeteScan was broken when I checked it).

Mudge contributed an article about testing supposed secure devices, which dovetails very nicely with Peter Guttman's paper on building a secure open source device for crypto-functions. I'll have more to say on that later. Sven Dietrich wrote about his own experiences working with Distributed Denial of Service software through his involvement in intrusion detection. And Carole Fennelly interviewed the conference's keynote speaker, Blaine Burnham, helping to explain exactly what Burnham plans to do in the future.

Burnham's speech struck several chords for me. There was the part about goatheads – very nasty, low-growing weeds endemic to the Southwest and the scourge of bicyclists. These weeds produce pretty little flowers, which turn into vicious spiked seeds quite capable of flattening any bike tire, as well as getting stuck in car tires (which is why they are found alongside roadways in many places).

Burnham used the goathead as an analogy. Bicycle riders have learned to take countermeasures (or have flat tires daily during the mid- to late summer), such as extra-thick tires, a plastic internal guard, or (in my case) green goo, anti-freeze mixed with fibers, that fills small holes quite nicely. The problem is, software vendors have yet to figure out about the green goo. When an exploit is discovered for NT or a CGI script, there is nothing that will automatically fix the problem. You, and your system, are history.

Burnham said that this is because no one is writing secure programs. Sure, everyone puts out patches, but that is hardly the proper solution when you need your server running, or when your company has become front-page news. Writing secure software will certainly help. But we have several years' experience with well-known problems with buggy code, for example buffer overflows, and yet buffer overflows are still uncovered at an alarming rate. (One day on BugTraq, a URL was posted for a site named LSD that had exploit code for 20 vulnerabilities alone.) Marcus Ranum has said before that writing secure code is not easy. And he has personal experience of this, not just because he was brave enough to teach a class in writing secure code, but also because his own code fell victim.

Was this the event that forever embittered Ranum? Just kidding, but I am guessing that it came close. Ranum had written large parts of the Firewall Toolkit (fwtk), only to fall prey to a buffer-overflow attack. The attack was not in his code, but in the use of the syslog() subroutine library called by his code for logging. More recently, WU-FTPD fell prey to problems involving logging using sprintf(), and setproctitle() also allegedly had a buffer overflow.

Burnham suggested that instead of just patching programs and trying to write secure code, we actually run secure operating systems. This idea dates back to research in the '60s and '70s, with ideas like the rings in MULTICS, or the concept of a security monitor. Not that we don't use rings in our operating systems today – just not well. For example, the Intel processor has four rings, but only two are used (see John Scott Robin's paper). UNIX, Linux, and NT systems all run OS code in the innermost, privileged ring, and everything else in ring 4 (or an outer ring on other processors). The problem with this is that an operating system, especially one where you can install drivers and loadable modules, is much too large to secure.

Instead, the core of an operating system should be the security monitor. This is very similar to a real microkernel system with a focus on security. (There are research versions of this design out there.) But the designs have so far proven to be too slow, and getting new device drivers for them is a serious problem. Still, I have written about this idea several times in the many years I have contributed to *;login:*, and years before that when I wrote for *UNIXWorld*. Our operating systems must be secure before we can expect our systems to be secure.

;login:

EDITORIAL STAFF

THEME ISSUE EDITOR
Rik Farrow <*rik@spirit.com*>

EDITORS
Tina Darmohray <*tmd@sage.org*>
Rob Kolstad <*kolstad@usenix.org*>

STANDARDS REPORT EDITOR
David Blackwood <*dave@usenix.org*>

MANAGING EDITOR
Jane-Ellen Long <*jel@usenix.org*>

COPY EDITOR
Eileen Cohen

TYPESETTER
Festina Lente

MEMBERSHIP AND PUBLICATIONS
USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: +1 510 528 8649
FAX: +1 510 548 5738
Email: <*office@usenix.org*>
WWW: <*http://www.usenix.org*>

This is not an unsolvable problem – just a very difficult one. One solution may be to build special hardware, perhaps a multiprocessing design where one processor handles device drivers, while another runs the secure OS, perhaps with virtual machines running insecure OSs above it. In this design the driver processor would not have access to system memory, and would rely on the security monitor for transferring data and commands to and from the driver processor and the main processor and memory. It could be done. The question is when.

I want to end this bit of musing by mentioning full disclosure. Full disclosure may disappear. That is, new vulnerabilities will not be announced, only software patches that might relate to security problems. This is exactly where we were six years ago, when some UNIX vendors (as well as a very large non-UNIX vendor) *never* posted information about security problems. They just didn't talk about it.

Today, we are at the opposite extreme. For example, on Labor Day, several different security vendors and teams announced that they had discovered serious problems (read, root compromise) via the locale mechanisms in glibc, right after several OS-distribution vendors announced patches for the problem. But the announcement was not simultaneous, so very large vendors, like Sun and HP, did not have their patches ready yet. PR through bug announcements is the current trend, and if the unruly mob doesn't learn some manners, we may soon find it gagged by law (or lawsuits).

The posting of complete, packaged exploits is another issue. On the one hand, I really appreciate having code to read, as that helps with my understanding of a problem (in UNIX at least – forget about the Win32 API). Unfortunately, it also helps the hoards of script kiddies, who will start trying to hump, er, exploit, every system with the appropriate port open, even if the architecture does not match.

I really do not want to see full disclosure go away. What I would like to see is some moderation, moderation that appears to be forthcoming from groups like SecurityFocus. SecurityFocus evolved out of Scott Chasin's BugTraq mailing list, which began after Brent Chapman got really upset when Chasin posted a sendmail root exploit to the old firewalls mailing list back in 1994. Chasin's posting was in response to a CERT advisory about sendmail that was so vague as to leave everyone wondering what the problem with sendmail might be. Chasin's post turned out to have nothing to do with the CERT advisory. (Read my article at <*http://www.spirit.com/Network/net0800.txt*> to learn more about this.)

Having enough information to determine that your systems are exploitable is good. Having a thousand script kiddies beating down your door is bad (very annoying, especially if you ask the Pentagon).

With that note, I wish you all secure operating systems, and a merry good year.

securing the DNS

by Evi Nemeth

Evi Nemeth is a member of the computer science faculty at the University of Colorado and a part-time researcher at CAIDA, the Cooperative Associationfor Internet Data Analysis at the San Diego Supercomputer Center. She is about to get out of the UNIX and networking worlds and explore the real world on a sailboat.

<evi@cs.colorado.edu>

As our society becomes more and more dependent on the Internet for information, communication, and business, the Domain Name System (DNS), which holds the Internet together, becomes a tasty target for hackers. A hacker who compromises DNS can divert all traffic destined for one host to another host without users ever knowing they have been led astray. The economic impact of such an attack can be huge.

Securing DNS does not stop Web sites from being broken into and defaced, but it does help to guarantee users that they are actually reaching the Web site they asked for. To be totally honest, securing the DNS will guarantee that you reach the correct IP address, but when connecting to this valid IP address, your connection might still be hijacked or mis-routed due to other weaknesses in the infrastructure. To really secure the Internet we need end-to-end authentication and encryption of the data sent over a connection. Securing the DNS via DNSSEC is the first step, as the DNS can then provide a way to distribute the keys required by any end-to-end security mechanisms. DNS is equally important for email, copying files, printing, or any application that uses domain names instead of network addresses.

There are two main points of vulnerability in the DNS system:

- Server-server updates
- Client-server communication

The Internet Software Consortium's BIND implementation addresses these vulnerabilities with separate mechanisms: TSIG/TKEY for server updates and DNSSEC for client-server lookups. The first allows pairs of servers, such as your master server and its slaves, to authenticate each other before exchanging data. TSIG/TKEY uses shared-secret-based cryptography. DNSSEC allows the client not only to authenticate the identity of a server but also to verify the integrity of the data received from that server. It uses public key cryptography. We describe each of these two mechanisms in detail and then look at some of the outstanding issues that are hampering the widespread deployment of secure DNS zones. Some of the material in this article is adapted with permission from Nemeth et al., *Unix System Administration Handbook, Third Edition*, Prentice Hall PTR, (in press). We assume that the reader is somewhat familiar with DNS resource records, the DNS naming hierarchy, `named` (the BIND name-server daemon), and its configuration file /etc/named.conf.

Securing DNS Transactions with TSIG and TKEY

While DNSSEC (covered in the next section) was being specified, the IETF developed a simpler mechanism called TSIG (RFC2845) to allow secure communication among servers through the use of transaction signatures. Access control based on transaction signatures is more secure than access control based on IP source addresses.

Transaction signatures use a symmetric encryption scheme. That is, the encryption key is the same as the decryption key. This single key is called a shared-secret key. You must use a different key for each pair of servers that want to communicate securely. TSIG is much less expensive computationally than public key cryptography, but it is only appropriate for a local network on which the number of pairs of communicating servers is small. It does not scale to the global Internet.

TSIG signatures sign DNS queries and responses to queries, rather than the authoritative DNS data itself, as is done with DNSSEC. TSIG is typically used for zone transfers between servers or for dynamic updates between a DHCP server and a DNS server.

Older versions of BIND do not understand signed messages and complain about them, sometimes to the point of refusing to load the zone.

TSIG signatures are checked at the time a packet is received and are then discarded; they are not cached and do not become part of the DNS data. Although the TSIG specification allows multiple encryption methods, BIND implements only one, the HMAC-MD5 algorithm.

BIND's dnssec-keygen utility generates a key for a pair of servers. For example, to generate a shared-secret key for two servers, serv1 and serv2, use the command

```
# dnssec-keygen -H 128 -h -n serv1-serv2
```

to create a 128-bit key and store it in the file Kserv1-serv2+157+00000.private. The file contains the string "Key:" followed by a base-64 encoding of the actual key.

The generated key is really just a long random number. You could generate the key manually by writing down an ASCII string of the right length and pretending that it's a base-64 encoding of something, or by using mimencode to encode a random string. The way you create the key is not important; it just has to exist on both machines.

Copy the key to both serv1 and serv2 with scp, or cut and paste it. Do not use telnet or ftp to copy the key; even internal networks may not be secure. The key must be included in both machines' named.conf files. Since named.conf is usually world-readable and keys should not be, put the key in a separate file that is included in named.conf. For example, you could put the snippet

```
key serv1-serv2 {
      algorithm hmac-md5 ;
      secret "shared-key-you-generated" ;
} ;
```

in the file serv1-serv2.key. The file should have mode 600 and its owner should be named's UID. In the named.conf file, you'd add the line

```
include "serv1-serv2.key"
```

near the top.

This part of the configuration simply defines the keys. To make them actually be used to sign and verify updates, each server needs to identify the other with a keys clause. For example, you might add the lines

```
server serv2's-IP-address {
      keys { serv1-serv2 ; } ;
} ;
```

to serv1's named.conf file and

```
server serv1's-IP-address {
      keys { serv1-serv2 ; } ;
} ;
```

to serv2's named.conf file. Any allow-query, allow-transfer, and allow-update clauses in the zone statement for the zone should also refer to the key. For example:

```
allow-transfer { key serv1-serv2 ;} ;
```

When you first start using transaction signatures, run named at debug level 1 (-d1) for a while to see any error messages that are generated. Older versions of BIND do not understand signed messages and complain about them, sometimes to the point of refusing to load the zone.

TKEY is an IETF protocol that BIND 9 implements to allow two hosts to generate a shared-secret key automatically without phone calls or secure copies to distribute the key. It uses an algorithm called the Diffie-Hellman key exchange, in which each side makes up a random number, does some math on it, and sends the result to the other side. Each side then mathematically combines its own number with the transmission it received to arrive at the same key. An eavesdropper might overhear the transmission but will be unable to reverse the math.

Securing Zone Data with DNSSEC

DNSSEC is a set of DNS extensions that authenticates the origin of zone data and verifies its integrity by using public key cryptography. That is, the extensions permit DNS clients to ask the questions, "Did this DNS data really come from the zone's owner?" and "Is this really the data sent by that owner?"

DNSSEC provides three distinct services: key distribution by means of KEY resource records stored in the zone files, origin verification for servers and data, and verification of the integrity of zone data. DNSSEC relies upon a cascading chain of trust: The root servers provide validation information for the top-level domains, the top-level domains provide validation information for the second-level domains, and so on.

Public key cryptosystems use two keys: one to encrypt (sign) and a different one to decrypt (verify). Publishers sign their data with a secret "private" key. Anyone can verify the validity of a signature with a matching "public" key that is widely distributed. If a public key correctly decrypts a zone file, then the zone must have been encrypted with the corresponding private key. The trick is to make sure that the public keys you use for verification are authentic. Public key systems allow one entity to sign the public key of another, thus vouching for the legitimacy of the key; hence the term "chain of trust."

Public key systems allow one entity to sign the public key of another, thus vouching for the legitimacy of the key; hence the term "chain of trust."

The data in a DNS zone is too voluminous to be encrypted with public key cryptography – the encryption would be too slow. Instead, since the data is not secret, a secure hash (e.g., an MD5 checksum) is run on the data and the results of the hash are signed (encrypted) by the zone's private key. The results of the hash are like a fingerprint of the data, and the signed fingerprint is called a digital signature.

Digital signatures are usually appended to the data they authenticate. To verify the signature, you decrypt it with the public key of the signer, run the data through the same secure hash algorithm, and compare the computed hash value with the decrypted hash value. If they match, you have authenticated the signer and verified the integrity of the data.

In the DNSSEC system, each zone has its own public and private keys. The private key signs each RRset (that is, each set of records of the same type for the same host). The public key verifies the signatures and is included in the zone's data in the form of a KEY resource record.

Parent zones sign their child zones' public keys. named verifies the authenticity of a child zone's KEY record by checking it against the parent zone's signature. To verify the authenticity of the parent zone's key, named can check the parent's parent, and so on back to the root. The public key for the root zone would be included in the root hints file.

Signing a Zone

Several steps are required to create and use signed zones. First, you generate a key pair for the zone. For example, in BIND 9,

```
# dnssec-keygen -a DSA -b 768 -n ZONE mydomain.com.
```

or in BIND 8,

```
# dnskeygen -D768 -z -n mydomain.com.
```

The table below shows the meanings of the arguments to these commands.

Argument	Meaning
For dnssec-keygen (BIND 9)	
-a DSA	Uses the DSA algorithm
-b 768	Creates a 768-bit key pair
-n ZONE mydomain.com.	Creates keys for a zone named mydomain.com
For dnskeygen (BIND 8)	
-D768	Uses the DSA algorithm, with a 768-bit key
-z	Creates a zone key
-n mydomain.com.	Creates keys for a zone named mydomain.com

dnssec-keygen and dnskeygen return the following output:

```
algorithm = 003
key identifier = 12345
flags = 16641
```

They also create files containing the public and private keys:

```
Kmydomain.com.+003+12345.key         # public
Kmydomain.com.+003+12345.private     # private key
```

Generating a key with dnssec-keygen can take a long time, especially if your operating system does not have /dev/random to help with generating randomness. It will ask you to type stuff and use parameters from your typing speed and pauses to get the randomness it needs. It might take five minutes, so don't get impatient, and keep typing until it stops echoing dots for each character you type.

The public key is typically $INCLUDEd into the zone file. It can go anywhere after the SOA record, but is usually the next record after SOA. The DNS key record from the .key file looks like this:

```
mydomain.com. IN KEY 256 3 3 BIT5WLkFva53IhvTBIqKrKVgme7...
```

where the actual key is about 420 characters long.

DNSSEC requires a chain of trust, so a zone's public key must be signed by its parent to be verifiably valid. BIND 8 had no mechanism to get a parent zone to sign a child zone's key other than out-of-band cooperation among administrators. BIND 9 provides a program called dnssec-makekeyset to help with this process.

dnssec-makekeyset bundles the keys you want signed (there may be more than just the zone key), a TTL for the resulting key set, and a signature validity period. For example, the command

```
# dnssec-makekeyset -t 3600 -s +0 -e now+864000
Kmydomain.com.+003+12345
```

bundles the public zone key that you just generated with a TTL of 3,600 seconds (one hour) and requests that the parent's signature be valid for ten days starting from now.

dnssec-makekeyset creates a single output file, mydomain.com.keyset. You must then send the file to the parent zone for signing. It contains the public key and signatures generated by the zone keys themselves so that the parent can verify the child's public key. Here is the mydomain.com.keyset file generated by the dnssec-makekeyset command above:

```
$ORIGIN .
$TTL 3600   ; 1 hour
mydomain.com        IN SIG      KEY 3 2 3600 20000917222654 (
                    20000907222654 64075 mydomain.com.
                    BE8V7nicLOARc0PRvBhBMeX7JXL3TdCUBY2Ah313pg+
                    Wq4THOq0U28Q= )
            KEY         256 3 3 (
                    BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCnV
                    57TBIeHkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJfboVma
                    8IGmONId2CSfryeuLdLLwW15bhhPHdw+nXWPFB7MY5s
                    bGLkokpuWmyHXkdWThr3A1ICWBs5GQRg8wMaIGOL4d
                    VSUWefQ/g4hGchEq12kieYVE4j9PE5p3uX2BNe0CIGNf05
                    c1VD6kYIn5Ip4hQZGwVL8hpi6NJsxp2U/krtS7GpHN55WA
                    fRY+joQ4AalY3f+AtapkGdV3lHjr1a7LG0qAgFAhNJ2jqKvoB
                    nXbWKKY9AlzMjsyleRdtRqn+V8vY30uTCkaaykrWhtu02QZ
                    pIGuwx294RudyA3gOQgR1aJ+X6BfUmXm2msmmHq//vL
                    mr )
```

In BIND 9, the parent zone uses the dnssec-signkey program to sign the bundled set of keys:

```
# dnssec-signkey mydomain.com.keyset Kcom.+003+56789
```

This command produces a file called mydomain.com.signedkey, which the parent (com) sends back to the child (mydomain.com) to be included in the zone files for mydomain.com. In BIND 8, the parent uses the dnssigner command.

The signedkey file is similar to the keyset file, except that the SIG record is associated with the com zone. Note, in our example, we generated the key for the com zone to use in the dnssec-signkey command; not the real one.

```
$ORIGIN .
$TTL 3600       ; 1 hour
mydomain.com            IN SIG   KEY 3 2 3600 20000917222654 (
                        2000090722265431263com.BAM/WIdPIwY6b4Aj8a5PZ
                        1UHwfo/qKl65HIlpitdvF2UgKaNJVEMSY4= )
                KEY         256 3 3 (
                        BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCn
                        V57TBIeHkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJ
                                                 ...
```

Once you have obtained the parent's signature, you are ready to sign the zone's actual data. Add the records from the signedkey file to the zone data before signing the zone. The signing operation takes a normal zone data file as input and adds SIG and NXT records immediately after every set of resource records. The SIG records are the actual signatures, and the NXT records support the signing of negative answers.

Here is a before and after example for our mydomain.com zone:

```
$TTL 3600       ; 1 hour
; start of authority record for fake mydomain.com
@    IN     SOA      mydomain.com. hostmaster.mydomain.com. (
              2000083000       ; Serial Number
              7200             ; Refresh - check every 2 hours for now
              1800             ; Retry   - 30 minutes
              604800           ; Expire  - 1 week (was 2 weeks)
              7200 )           ; Minimum - 2 hours for now

              KEY   256   3   3 (
                    BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCnV57TBIe
                    HkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJfboVma8IGmONId2
                    CSfryeuLdLLwW15bhhPHdw+nXWPFB7MY5sbGLkokpuWmyH
                    XkdWThr3A1ICWBs5GQRg8wMaIGOL4dVSUWefQ/g4hGchEq1
                    2kieYVE4j9PE5p3uX2BNe0CIGNf05c1VD6kYIn5Ip4hQZGwVL8h
                    pi6NJsxp2U/krtS7GpHN55WAfRY+joQ4AalY3f+AtapkGdV3lHjr1
                    a7LG0qAgFAhNJ2jqKvoBnXbWKKY9AlzMjsyleRdtRqn+V8vY30u
                    TCkaaykrWhtu02QZpIGuwx294RudyA3gOQgR1aJ+X6BfUmXm
                    2msmmHq//vLmr )

              IN    A        128.138.243.151
              IN    NS       @
              IN    NS       anchor
              IN    NS       zamboni
              IN    MX       10    @
              IN    MX       30    anchor
              IN    LOC      40 00 23.5 N 105 15 49.2 W 1900m

localhost                       IN        A        127.1
anchor                          IN        A        128.138.242.1
                                IN        A        128.138.243.140
                                IN        MX       10  anchor
                                IN        MX       99  @
awesome                         IN        A        128.138.236.20
                                IN        MX       10  awesome
                                IN        MX       99  @
zamboni                         IN        A        128.138.199.7
                                IN        A        128.138.243.138
                                IN        MX       10  zamboni
                                IN        MX       99  @
```

In BIND 8, you use the dnssigner program in the contrib directory of the distribution to sign a zone; in BIND9, you use the dnssec-signzone command. For example, the command

```
# dnssec-signzone mydomain.com Kmydomain.com.+003+12345 # BIND 9
```

reads the zone file mydomain.com and produces a signed version using the private key in the Kmydomain.com+003+12345.private file. The resulting file is called mydomain.com.signed. If you forget to include the key file on the command line, you get an obscure error message about an "inappropriate ioctl for device" from the module entropy.c.

It can take a long time to sign a zone, especially if your system does not have /dev/random, because it asks you to type a few sentences for each signature it generates. Quite a pain after a while. If you try to fool it by cutting and pasting text in, it makes you type more till it feels there has been sufficient randomness generated. Here is a portion of the resulting signed zone file for mydomain.com:

```
$ORIGIN .
$TTL 3600      ; 1 hour
mydomain.com             IN SOA  mydomain.com. hostmaster.mydomain.com. (
                         2000083000  ; serial
                         7200        ; refresh (2 hours)
                         1800        ; retry (30 minutes)
                         604800      ; expire (1 week)
                         7200        ; minimum (2 hours)
                         )
               SIG          SOA 3 2 3600 20001008023531 (
                         2000090802353164075mydomain.com.BFN/8mIRX/M
                         W01kMoe+7qId63LB7Tbb9t98/NnfY16WQgItk03FDXTk
                         = )
               NS           mydomain.com.
               NS           anchor.mydomain.com.
               NS           zamboni.mydomain.com.
               SIG          NS 3 2 3600 20001008023531 (
                         20000908023531 64075 mydomain.com.
                         BAkrse9uTdANxbGA0kaWkjiippeCCUBcvHGR7zDOt+k
                         STeGbVfJy8iw= )
               SIG          LOC 3 2 3600 20001008023531 (
                         20000908023531 64075 mydomain.com.
                         BIARXt5zqiPy08Ca7T7AiUCau1PJEWlv3uHTQci0f3g5nlr
                         kw1exaqM= )
               SIG          MX 3 2 3600 20001008023531 (
                         2000090802353164075mydomain.com.
                         BEHMocIH/p1cL0FlQTz1cfEZzqHHHf1BBLSy2FtU1H6v
                         5DXZy9zkyOw= )
               SIG          A 3 2 3600 20001008023531 (
                         20000908023531 64075 mydomain.com.
                         BGKrpBrAkCtHcuzX57heH5sS0MYnFRC3MqeRMf3i881
                         Y3ZD+Q+E9r24= )
               SIG          KEY 3 2 3600 20000917222654 (
                         20000907222654 31263 com.
                         BAM/WIdPIwY6b4Aj8a5PZ1UHwfo/qKl65HIlpitdvF2UgK
                         aNJVEMSY4= )
$TTL 7200      ; 2 hours
               SIG          NXT 3 2 7200 20001008023531 (
                         20000908023531 64075 mydomain.com.
                         BDvw+QgYBcmIXeS4qMyMNDtB8K+sX5Jb2zKMRCcQ
                         4uFySJJVQ/s6A1w= )
               NXT          anchor.mydomain.com. ( A NS SOA MX SIG
                            KEY LOC N XT )
$TTL 3600      ; 1 hour
               KEY          256 3 3 (
                         BIT5WLkFva53IhvTBIqKrKVgme7r/tbnBTRkLDDKjGYCn
                         V57TBIeHkZSgbJ7jfYtuTLv4a2OIF5jJDoHD8LEFKNJfbo
                         Vma8IGmONId2CSfryeuLdLLwW15bhhPHdw+nXWPFB
                         7MY5sbGLkokpuWmyHXkdWThr3A1ICWBs5GQRg8w
                         MaIGOL4dVSUWefQ/g4hGchEq12kieYVE4j9PE5p3uX2
                         BNe0CIGNf05c1VD6kYIn5Ip4hQZGwVL8hpi6NJsxp2U/
                         krtS7GpHN55WAfRY+joQ4AalY3f+AtapkGdV3lHjr1a7L
                         G0qAgFAhNJ2jqKvoBnXbWKKY9AlzMjsyleRdtRqn+V8v
                         Y30uTCkaaykrWhtu02QZpIGuwx294RudyA3gOQgR1aJ
                         +X6BfUmXm2msmmHq//vLmr )
```

```
                A         128.138.243.151
                MX        10 mydomain.com.
                MX        30 anchor.mydomain.com.
                LOC       40 0 23.500 N 105 15 49.200 W 1900.00m 1m
                          10000m 10m
$ORIGIN mydomain.com.
anchor          SIG       MX 3 3 3600 20001008023531 (
                        20000908023531 64075 mydomain.com.
                        BFEtOCT+y0dQPx7Am7gpxD9SjEl+USuaE7qExUOrX22
                        X7wjqJFJbqdo= )
                SIG       A 3 3 3600 20001008023531 (
                        20000908023531 64075 mydomain.com.
                        BDwfBm2j6xFLoXttzvtuln9ZD+9qUWBAwSBJVB06WJ/
                        Rc6+F1ubj/fs= )
$TTL 7200       ; 2 hours
                SIG       NXT 3 3 7200 20001008023531 (
                        20000908023531 64075 mydomain.com.
                        BIMwxryI8NyfWupBe4JJmeRCCj1/FnyPjxAuBOQKTRX
                        X4FsaDrma1X4= )
                NXT       awesome ( A MX SIG NXT )
$TTL 3600       ; 1 hour
                A         128.138.242.1
                A         128.138.243.140
                MX        10 anchor
                MX        99 mydomain.com.
                                    …
```

The signedkey file from the parent domain .com gets slurped into the processing if it is in the same directory as the zone file being signed (see the SIG KEY record associated with com toward the top of the example). There is quite an increase in the size of the zone file, roughly a factor of three in our example. The records are also reordered.

A SIG record contains a wealth of information:

- The type of record set being signed
- The signature algorithm used (in our case, it's 3, the DSA algorithm)
- The TTL of the record set that was signed
- The time the signature expires (as yyyymmddhhssss)
- The time the record set was signed (also yyyymmddhhssss)
- The key identifier (in our case 12345)
- The signer's name (mydomain.com.)
- And finally, the digital signature itself

To use the signed zone, change the file parameter in the named.conf zone statement for mydomain.com to point at mydomain.com.signed instead of mydomain.com. In BIND 8, you must also include a pubkey statement in the zone statement; BIND 8 verifies the zone data as it loads and so must know the key beforehand. BIND 9 does not perform this verification. It gets the public key from the KEY record in the zone data and does not need any other configuration.

Whew! That's it.

Negative Answers

Digital signatures are fine for positive answers like "Here is the IP address for the host anchor.mydomain.com, along with a signature to prove that it really came from mydomain.com and that the data is valid." But what about negative answers like "No such host"? Such negative responses typically do not return any signable records.

In DNSSEC, this problem is handled by NXT records that list the next record in the zone in a canonical sorted order. If the next record after anchor in mydomain.com is awesome.mydomain.com and a query for anthill.mydomain.com arrived, the response would be a signed NXT record such as

```
anchor.mydomain.com. IN  NXT  awesome.mydomain.com ( A MX SIG NXT )
```

This record says that the name immediately after anchor in the mydomain.com zone is awesome, and that anchor has at least one A record, MX record, SIG record, and NXT record. The last NXT record in a zone wraps around to the first host in the zone. For example, the NXT record for zamboni.mydomain.com points back to the first record, that of mydomain.com itself:

```
zamboni.mydomain.com. IN  NXT  mydomain.com. ( A MX SIG NXT )
```

NXT records are also returned if the host exists but the record type queried for does not exist. For example, if the query was for a LOC record for anchor, anchor's same NXT record would be returned and would show only A, MX, SIG, and NXT records.

We have described DNSSEC as of BIND v9.0.0rc5 (September 2000). Judging from the significant changes that occurred during the beta cycle, this information may not be current for long. As always, consult the documentation that comes with BIND for the exact details.

Outstanding Issues

Now that we have described the two mechanisms available in BIND for securing the DNS, let's look at some of the potential problems.

CACHING AND FORWARDING

DNSSEC is at odds with the notions of caching and forwarders.

DNSSEC is at odds with the notions of caching and forwarders. DNSSEC assumes that queries contact the root zone first and then follow referrals down the domain chain to get an answer. Each signed zone signs its children's keys, and the chain of trust is unbroken and verifiable. When you use a forwarder, however, the initial query is diverted from the root zone and sent to your forwarding server for processing. A caching server that is querying through a forwarder will recheck signatures, so responses are guaranteed to be secure. But, for the query to succeed, the forwarder must be capable of returning all the SIGs and KEYs needed for the signature checking. Non-DNSSEC servers don't know to do this, and the RFCs ignore the whole issue of forwarding.

BIND 9 implements some extra features beyond those required by RFC2535 so that a BIND 9 caching server can use DNSSEC through a BIND 9 forwarder. If you are using forwarders and want to use DNSSEC, you might have to run BIND 9 throughout your site.

Unfortunately, those busy sites that use forwarders and caching are probably the sites most interested in DNSSEC. Alas, the standards writers didn't quite think through all of the implications for the other parts of the DNS system.

PUBLIC KEY INFRASTRUCTURE

DNSSEC also relies on the existence of a public key infrastructure that isn't quite a reality yet. There is no smooth way to get the parent to sign a child's keys; we cannot yet send mail to the hostmaster@com and get signed keys back. A public key infrastructure

NLnet Labs is currently running an experiment on DNSSEC issues in a special domain called NL.NL. They are building tools to automate the signing of subdomain keys and helping people get their domains secured.

is needed for other applications too, such as IPSec (a secure version of the IP protocol) or e-commerce. DNSSEC is the first step in the chain of a series of security enhancements being deployed on the Internet.

The private key for the root zone is essential for the whole process to work. It must be used whenever the root zone changes and needs to be re-signed. Therefore it must be accessible. What do we do if the private root key is lost or stolen? Generating a new key pair is fast, but distributing it to millions of DNS servers isn't. How do we change keys? There must be some period of time during which both the old key and the new one are valid if caching is to work. How do we plan for changing important keys – the root, the key for the COM zone, etc.? How do we engineer the switchover to not destabilize the network? If the root key is built into the software, then a compromise implies that every DNS server out there must be manually touched and changed. The disruption to the network would be worse than the damage that whoever stole the root key could do.

SIGNING BIG ZONES

The COM zone is over 2GB. It is typically updated twice a day. But with current hardware and software-signing the COM zone takes several days. An incremental mechanism for re-signing a zone is built into the current BINDv9 distribution. Re-signing, when not much has changed, takes about 5% of the time to do the original signing. We are within striking distance of being able to maintain a signed copy of the COM zone.

The folks at NLnet Labs (<*http://www.nlnetlabs.nl/dnssec*>) have experimented with signing the top-level DE, NL, ORG, and COM zones. Some of their results are shown below:

DE	full zone	13 hours	FreeBSD 3.4 PC
DE	delegation zone	4 hours	FreeBSD 3.4 PC
ORG	full zone	42 hours*	Red Hat Linux 6.2, DEC/Compaq Alpha
COM	delegation zone	50 hours	Red Hat Linux 6.2, DEC/Compaq Alpha

* It took 2 hours to re-sign after 1 record changed

The COM zone snapshot included 12 million delegations and was sorted before signing (three hours with the standard UNIX `sort` command). The process used 9GB of virtual memory, and it took an additional nine hours to write the signed zone out to disk.

NLnet Labs is currently running an experiment on DNSSEC issues in a special domain called NL.NL. They are building tools to automate the signing of subdomain keys and helping people get their domains secured.

PERFORMANCE

Signed zones are bigger. Signed answers to queries are bigger. UDP, the default transport protocol used by DNS, has a limitation that makes the maximum packet size 512 bytes in the default case. If an answer is greater than 512 bytes, a truncated answer comes back in a UDP packet and the client re-asks the query using TCP. TCP is slower and requires more network traffic. It's unclear whether the current servers for COM could keep up with the query rate if a large portion of the DNS traffic were TCP.

Verifying signatures also costs CPU time and memory; the cost is about one-twentieth of the cost of signing. The actual rates for both signing and verifying depend on the encryption algorithm used, with DSA-512 the fastest (signing about 135 domains/sec. on a 500Mhz FreeBSD PC) and RSA-1024 the slowest (17 domains/sec.). DSA-768, a popular algorithm, is in the middle at 62 domains/sec.

Transaction signatures (TSIG/TKEY) use less CPU time and network bandwidth than do public key authentication methods, but they guarantee only that you know where your responses came from, not that the responses are correct. A combination of a TSIG relationship with a server known to do full DNSSEC might provide a reasonable degree of security. It is not possible to have a TSIG relationship with every server you might ever want to talk to, since TSIG relationships must be manually configured.

Conclusions

The ISC BIND version 9 contains an initial set of tools for sites to begin securing their DNS. However, the public key infrastructure and automated mechanisms for a child zone to have its key signed by its parent are not yet in place. Look for DNSSEC to be fully deployable in the near future – it is the key ingredient in a public key infrastructure that can be used by any application requiring authentication, security, or privacy. Folks with very sensitive data (banks, e-commerce sites, military installations, etc.) might want to start experimenting with DNSSEC now, at least within the corporate intranet.

Look for DNSSEC to be fully deployable in the near future.

repeatable security

by David Brumley

David Brumley is the assistant computer security officer for Stanford University and a consultant with Securify, Inc. David also runs the white-hat security site www.theorygroup.com.

<dbrumley@stanford.edu>

Repeatable Process

After a computer security policy is written, the real work begins – implementing it! Implementation is the process of converting a written policy into a set of specific procedures. Implementation requires the translation of policy statements into current technology on current hardware, an often arduous task.

Implementation is difficult, primarily because picking the right technology involves tradeoffs. One product may streamline a business process yet create numerous security risks. Implementors must decide whether the cost of risk mitigation is less or more than the savings incurred by the software.

To make matters more difficult, current products emphasize features over economy of mechanism. Economy of mechanism gives a clear method for a solution, while features tend to cloud which mechanism is appropriate. Balancing the two is difficult. For example, some surveys show that over 300 dialog boxes must be answered to correctly install and configure Microsoft Windows NT 4.0. Because of the sheer number of mechanisms, implementors may have an incomplete understanding of all the tradeoffs being made.

Ultimately, every organization must decide which technologies it supports and which it doesn't. Sadly, many organizations stop there. Each piece of software supported should also have a supported configuration. The reason: Any piece of software may have thousands of switches and dialog boxes that, when configured differently, create radically different solutions. For example, Windows NT 4.0 Workstation out of the box is very different from the same software configured with C2 security.

Instead, organizations rely upon software installed by hand in an ad hoc fashion. Predictably, error occurs. However, there is a better way.

Cloning a computer can be defined as the process of taking an installed system and duplicating the configuration across many hosts in a repeatable and automatic fashion. By definition, cloning is a way of managing the risk of human error. While cloning does not mitigate the risk of incorrect policy implementation, it does assure that the time and thought spent on a correct implementation is not wasted by the introduction of human errors.

In other words, if an implementation of a policy is correct, cloning ensures that each system cloned adheres to exactly the same standards. Cloning takes the traditionally ad hoc method of manual installation and turns it into a repeatable process with repeatable results. Cloning adds a development cycle to workstation installation and management. As a direct consequence, cloning gives the benefits of a true development cycle.

A clonable image is called a source image. There are two main methods for creating a source image. The first is to install a source host exactly as you want and then duplicate it. The second mechanism is to create your own distribution with all the configuration details self-contained.

An example of the first method would be the Norton Ghost product. Norton Ghost can be used to clone WinTel machines. The first step to using Norton Ghost (and products like it) is to install a source host. That source machine is then loaded onto a distribution server. Machines that you wish to clone contact the distribution server and download the image straight to disk.

RPM-based Linux, such as Red Hat, is a good example of creating your own distribution. You choose which RPMs (Red Hat packages) you wish to install and include them in a certain ftp/nfs directory. Then, when a client wishes to use the distribution, it simply FTPs the image to disk.

Often the first mechanism can be recognized because everything except plug-and-play hardware must be the same between source and destination. This is expected, since the system is simply copied over from the source to the destination without any additional drivers. The second method allows for different types of hardware between source and destination, but is not readily available for all platforms.

Time Saved

What are other reasons to clone machines? Cloning saves time. To illustrate, imagine the classic situation where an administrator must install 100 machines. On each machine, the administrator inserts the system CD, boots the computer, then manually answers each dialog question. Even while the administrator is not answering installation questions, he or she cannot stray far from the computer.

The total time it takes the administrator to install the 100 hosts is equal to the amount of time to install one host times 100. In computer-science notation, we would say the task of completing the installations is accomplished in O(N) time. Now it's unusual for a computer administrator to install 100 systems in a single day, so often the time spent goes unnoticed. However, as the saying goes, you can "nickel and dime" your time away. Time really adds up when installing a few today, a few tomorrow, and ten next week.

With cloning, time is saved by decreasing the total time spent when the number of hosts is increased. This concept is easiest to understand graphically. In Figure 1, the dotted line is the time it takes to manually install computers. The solid line is the time it takes to clone computers. Installing only one system takes less time than cloning, simply because there is a small cost associated with setting up the cloning mechanism. However, the mechanism needs to be set up only once. After installing only a few systems, this ramp-up cost becomes negligible, and cloning becomes profitable.

Normally, cloning involves an entire operating system plus any necessary applications. Which operating systems can be cloned? Almost every modern OS has some sort of built-in cloning capability. Windows NT/2000 has the Remote Installation Service (RIS). RPM-based Linux systems allow easy creation of custom distributions. IRIX has a tool named RoboInst. Solaris has JumpStart. The list goes on and on.

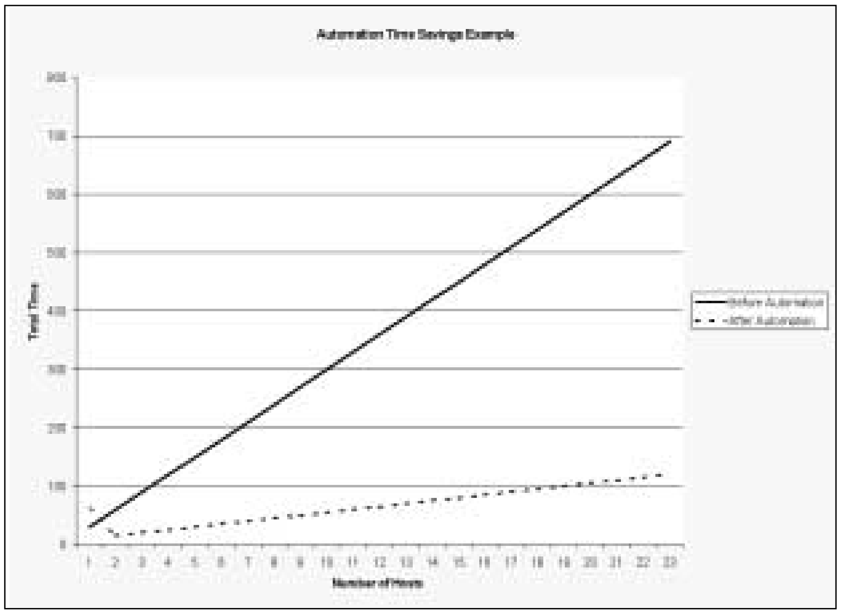


Figure 1

Regardless of the specific cloning mechanism, the greater the attention paid to planning your source image (that which you clone from), the greater the benefits. However, the things that make or break a project are how critically you think about incorporating secure authentication, remote administration capability, system security, and productivity applications into your distribution.

Time saved may not be the only advantage of cloning; you may save your organization from a lawsuit!

Authentication and Administration

Providing secure authentication is a necessity in today's hostile Internet. To manage risk, employers must not only ensure that employees' passwords are safe, but must also provide an audit trail of authentication events. Quite often, large lawsuits have been avoided because companies can show that both proper and appropriate action was taken, which can be proven with good authentication data. Time saved may not be the only advantage of cloning; you may save your organization from a lawsuit!

Remote administration allows for automatic updates and troubleshooting of a machine. Typical examples are sudo, Kerberos, and PC Anywhere. To maximize benefits, the remote administration mechanism should leverage off a single sign-on infrastructure. For example, at Stanford University we install Kerberos on each public cluster machine. Kerberos provides encrypted authentication. Kerberos also provides a mechanism for listing principals that can log into an account. We set up our public cluster machines to allow the Kerberos principal "dbrumley.root'" to log in to the root account.

In other words, I can use my active authentication credentials for authorization into various accounts. (This demonstrates a very good reason for distinguishing between authorization and authentication.) Since Kerberos is single sign-on, I can script administration commands to multiple machines. For example, here is a tcsh script to print out the date on each machine:

```
# cat stanford_hosts.list
  elaine1.stanford.edu
  elaine2.stanford.edu
  elaine3.stanford.edu
  # foreach host (`cat stanford_hosts.list`)
  > echo $host
  > /usr/kerberos/bin/rsh $host date
  > end
  elaine1.stanford.edu
  Sun Aug 20 10:14:10 PDT 2000
  elaine2.stanford.edu
  Sun Aug 20 10:14:10 PDT 2000
  elaine3.stanford.edu
  Sun Aug 20 10:14:11 PDT 2000
```

Notice how the date command above shows it took only one second to execute a command on three machines. More complex scripts can be created, such as mounting a patch tree and installing it. For example:

```
# foreach host (`cat stanford_hosts.list`)
> echo $host
> /usr/kerberos/bin/rsh $host ''mount genesis:/export/home /mnt; cd
/mnt/patches; ./install.sh; umount /mnt;''
> end
```

Stanford uses Kerberos not only for secure single sign-on, but also as a remote administration tool. With Kerberos, a handful of administrators can administer several hundred hosts each with about 30,000 active accounts!

OS Hardening

In their classic book *Firewalls and Internet Security,* Cheswick and Bellovin state as an axiom of computer security that all programs are buggy. A direct corollary is that if you don't run a program, it doesn't matter if it is buggy. More narrowly, with a UNIX-type system it doesn't matter whether a program is buggy or not if the program never executes with elevated privileges.

Operating System (OS) hardening is primarily concerned with reducing the number of programs that run with elevated privileges, such as network services and setuid programs. A typical hardening script will turn off unneeded services and unused setuid programs to mitigate the risk of exploitation. A side benefit is that if a program is not used, it doesn't need to be patched!

During cloning, it is important that your source image be hardened against attack. All services that are not normally needed should be turned off. A common mistake is to leave services enabled on your source image that are needed only by a small subset of cloned machines. It is much better to disable the services on all systems and manually reenable them when needed. The reason is twofold. First, if you do not need a service on the majority of the systems, you spend more time disabling it on the majority of systems than if you simply enabled it on the few where it was needed. It's just a matter of simple arithmetic. Second, services left on have a tendency to stay on simply because of human error, procrastination, or lack of time.

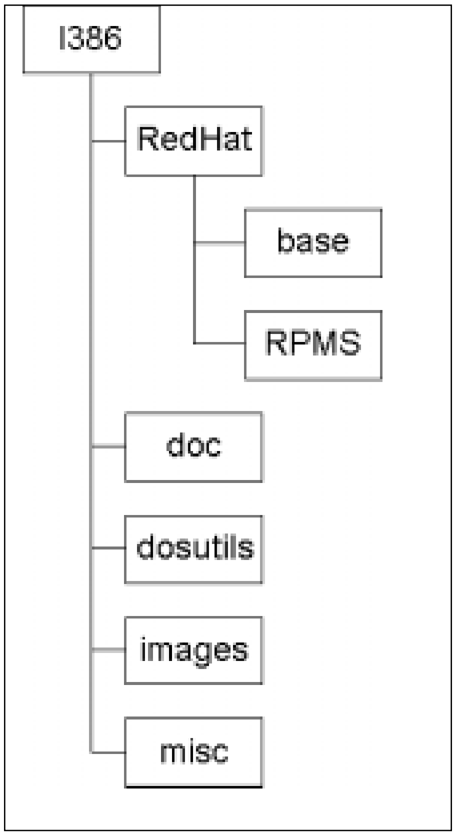
What if you do not know whether a given service or program needs privileges? One of my axioms of computer security states that if you don't know what it does, you shouldn't be running it. There is a wealth of tools to help you find out what a program does and what it is used for. Instead of simply ignoring the problem, read the man page, ask questions, and do traces of the program before installing it into a source image or production system.

An Example: Creating Your Own Red Hat Distribution

To emphasize how easy it is to make your own distribution, here are the steps needed to create your own Red Hat distribution:

1. Mirror Red Hat.
2. Create your own customization packages.
3. Include your packages in the distribution, remove packages not needed.
4. Inform the installation mechanism of your new package lists.
5. Install away.

Step 1 is to mirror Red Hat Linux. You can either sign up with Red Hat to become an official mirror site, or you can ask one of the primary mirrors if you can mirror off of them. The important thing is to download the directory tree for each version of Red Hat you are going to support.



Directory Tree

The i386 directory is the start of the distribution for the x86 architecture. If you like, the same techniques will carry over to the SPARC and PPC directory trees.

Underneath the i386 directory are images, dosutils, Red Hat, doc, and misc. The images directory is where the boot images are kept. dosutils contains programs like rawrite.exe and fips.exe that help users install Linux from a MS Windows system. doc is self-explanatory. misc is an interesting directory, as it contains the source code for the boot and second images. However, there is no need to rebuild the images unless you want to change the verbiage (or something even more drastic) seen during installation. The Red Hat directory contains all the information needed after the initial boot disk to install and configure Red Hat Linux, which we will explore later.

Step 2 is to create your own packages. There are several books and HOWTOs that describe this process. Ed Bailey's *Maximum RPM* from Red Hat is a good starting place to learn about building your own packages. However, it is a bit outdated, so be sure to consult the online manual pages for possible changes.

There are a few tricks to building successful RPMs for a distribution. The first is that RPMs are installed in a pseudo-alphabetical order. Therefore, if there is an RPM that must run first or last, it's important to name it correctly. For example, my OS-hardening RPM is named "zzsecurity" because it turns off services and disables setuid programs – things I want last during installation to avoid them being overwritten.

Second, I've found it useful to keep custom configuration options in separate RPMs instead of editing the ones bundled by Red Hat. I do this primarily for maintenance reasons; it makes it easy to identify which RPMs I provide and which are part of the standard Red Hat distribution.

Step 3 is to include your RPMs into the standard distribution. This is done by editing the Red Hat components file `i386/Red Hat/base/comps`. The `comps` file is a flat text file, with the format:

```
<Component File Format Version>
blank line
<Component 1>
blank line
<Component 2>
blank line
....
EOF
```

If you don't see a component already listed where your RPMs fit, you can create your own component group. The format for each component is:

```
(0|1) (—hide)? <name> {
name1.rpm
name2.rpm
name3.rpm
}
```

Choose either 0 or 1, depending on whether or not you want the package selected by default under custom installation. Also, note that the name of the component is completely arbitrary. For example, Stanford has one called "Stanford," which looks like:

```
1 Stanford {
zzsecurity.i386.rpm
libsafe.i386.rpm
afs.i386.rpm
kerberos.i386.rpm
}
```

If you define your own component, you can use that in later components to make it part of the standard options. For example, to include everything from the "Stanford" component into the "Workstation" component, simply add the name "Stanford" to the workstation component list.

Step 4 is the easiest. When using the Red Hat installer, a database is kept of RPM dependencies, size, and other information. That information is used by the installer to make sure all prerequisites and dependencies are installed properly. After you build your own RPMs and incorporate them into the component list, that database needs updating. When run from the i386 directory, i386/misc/src/anaconda/utils/genhdlist will rebuild the database for you. Note that the genhdlist command may be different between Red Hat versions, so use the genhdlist included with each version.

Lastly, you should test your distribution. During testing, you are preparing to "go live" with a new environment. Plans for support and maintenance should be in place before deploying sitewide. You'll want to support your Red Hat distribution the same way you would support other software: with a bug repository, a Web page describing basic installation, and so on.

For those who want a working example of a distribution, I've put up all the scripts and RPMs for my distribution at <*http://www.theorygroup.com/Tools/TGLinux*>. TGLinux is based upon a distribution I did for Stanford University that has been successfully installed on several thousand systems.

Branding

When creating a Red Hat distribution, there are several ways to do "lite branding." Here is a short list of ideas:

1. Change the graphical login logo to your own. It's located at <*usr/share/pixmaps/redhat/redhat-transparent.png*>.

2. Incorporate the latest fixes and patches into your distribution nightly. An example script can be found at: <*http://www.theorygroup.com/Tools/TGLinux/scripts/merge.pl*>.

3. Place a common motd and banner in /etc/issue and /etc/issue.net. Note that these files are automatically recreated at boot from /etc/init.d/rc.local normally, so you may have to make some additional changes to that startup script.

4. Burn CD-ROMs of the distribution for home users.

Summary

Although the cloning mechanism may change for other operating systems, certain tenets always apply. First, by creating your own clonable image, you have the opportunity to deploy and enforce your security policy. Second, cloning saves time. Third, cloning lends itself to a true development cycle with all its benefits.

But just as with everything else, the more thought put into planning, the better the results. Too often, we find ourselves typing in the same thing day after day. To have computers do what they should – enhance productivity – anything you find yourself doing multiple times should be automated. By automating tasks, you will create a repeatable process with repeatable results, an utter necessity to compete in the upcoming e-business world and participate in the hostile Internet.

To have computers do what they should – enhance productivity – anything you find yourself doing multiple times should be automated.

correlating log file entries

by Steve Romig

Steve Romig is in charge of the Ohio State University Incident Response Team, which provides incident response assistance, training, consulting, and security auditing.

<romig@net.ohio-state.edu>

I have often needed to peruse log files from different systems while investigating computer crime, performance issues, and other odd happenings – and I've learned a few tricks that I'd like to share with you. The general principles will apply to most investigations, but I'll draw my examples mostly from the UNIX and incident-response worlds with which I am most familiar.

I've written most of this while sitting at Camp Ohio, a 4-H camp, where I'm volunteering as a counselor for my church's junior high summer camp. Trying to write an article on such a technical subject between archery and setting up a campfire and night time zip line is an interesting challenge (a zip line is where you jump off a tower suspended below a cable by pulley and harness and ride the cable down to the ground some distance away – imagine doing that in the dark!) Between my co-counselor Marco and myself we had more computing power with us than the rest of the camp combined, but amazingly our cabin still didn't win the "geekiest cabin" award the day that was the theme for cabin clean-up. Maybe if we had had a working Internet connection . . .

Let's suppose that you are investigating a compromised computer, and you are fortunate enough to have tracked the activity back to the source and have access to all of the systems involved. In our case, a suspect used his home computer to connect to the Internet through our modem pool using a stolen account. Once on the Internet, he used a variety of tools to probe for and break into victim hosts for various purposes. (See Figure 1.)

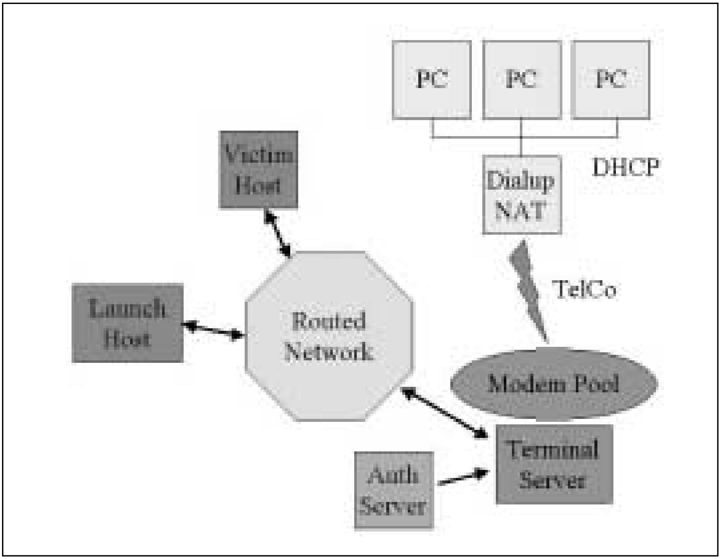


Figure 1

One common goal in these sorts of investigations is to reconstruct a chronological record of events and a list of other facts. Once we have done that, we develop one or more theories that account for this history and set of facts. If we are working on the side of the prosecution in a computer-crime investigation, our prime theory would be something along the lines of "the butler did it with mstream in the kitchen." If we are working on the side of the defense, our theory might be "the prosecution's theory didn't account for this and that evidence that shows that the butler couldn't have done it." The supporting evidence and these theories are presented before the court and the jury ("the trier of fact") is called upon to determine whether the prosecution has sufficiently proven its case or not. Obviously, how well we can construct the record of events and fit the pieces together has great bearing on the outcome of the investigation.

We need to consider several issues. First, we need to be proficient at finding the evidence. If you can't find the evidence in the first place, you'll have a hard time fitting it into your reconstructed chain of events. We also need to understand what the evidence actually means. If we misunderstand the evidence, then either our reconstruction will be wrong or we'll create faulty theories that explain the evidence. Finally, we need to understand how to piece evidence from different sources together to create a cohesive reconstruction. If we know where the evidence can be found, what it means, and how it fits together, then we'll be well on our way to reconstructing the chain of events. Note that I am totally ignoring issues concerning preservation of evidence for use in a civil or criminal trial. Sorry!

Know Where the Evidence Is

I won't dwell on this here – full treatment of the subject is way beyond the scope of this short article. In general, this means that you have to know where evidence pertaining to your case *might* be, and then look to see whether you can actually find it. For instance, in our example investigation, we might find evidence in the following locations (look back at Figure 1):

Home system	Dial scripts, dial logs, files containing output from exploit tools, lists of compromised hosts, etc.
Phone system	Phone traces or pen registers
Modem pool	TACACS, TACACS+, or RADIUS authentication logs
Networks	logs of network activity, such as Cisco Netflow logs or from the use of tools like Argus
Victim and intermediate hosts	Syslog records showing access to network services through TCP wrappers or other means; login records such as utmp, wtmp, wtmpx (or in syslog if you are smart enough to use loginlog, a program that transcribes wtmp entries to syslog); processes running on the system (and the associated memory, binaries, network connections, and files); free and slack space on the filesystem, and so on.

Think about the components involved in the incidents you are investigating – what information might they contain? If you don't know enough about them, it doesn't hurt to find an expert and ask questions. Many people fail in their investigations because they fail to ask questions about the components involved and thereby miss important evidence.

What the Evidence Means

It is relatively easy to understand where the evidence might lie. Draw a block diagram of the system under investigation and consider each component in turn – that at least gets you a high level view. Understanding what the evidence actually means is trickier. For one thing, it involves a deeper understanding of the component systems involved. At the very least, we need to understand how the evidence is created or compiled – for instance, knowing that the UNIX `login` program (and some others, like `sshd`) updates the `wtmp/wtmpx/utmp` logs and under what circumstances.

Knowing what the evidence means helps us avoid conclusions that aren't logically supported by the evidence. For example (and pardon me if this seems simplistic), a TACACS log entry that indicates that the "romig" account logged in means just that – the "romig" account was used to log in. It does not prove that the owner of the account was the one who used the account to log in, although the theory that "Steve Romig, the owner of the romig account, used it to log in at this time" is consistent with this evidence. Similarly, a DHCP (Dynamic Host Configuration Protocol) server log that shows that a host with a particular MAC address had a lease for a given IP address does not mean that that host was the only host using that IP address during that time period; it just means that this host held the lease. The theory that "this host held the lease for this IP address at the time and used that address to probe the victim" is consistent with the lease evidence, but the lease evidence doesn't conclusively prove this theory, since there are other plausible theories that are also consistent with this evidence.

Understanding what the evidence means also helps us recognize potential blind spots. One modem pool that I worked with used a pair of authentication servers handling authentication requests in a round-robin fashion. This meant that log entries pertaining to login/logout events for any given terminal server port could be found in the logs from either server. If we only looked at the records from authentication server A (see Figure 2), we might mistakenly conclude that the "romig" account was used to authenticate the session that spans 1:15:21 (the time that some nefarious Internet crime

time	authentication server A	authentication server B
1:02:12	login - romig	
1:10:32		logout
1:10:56		login - farrow
1:26:09	logout	

Figure 2: Login/logout events for a single port on a terminal server.

occurred, which we traced back to this terminal server port). Note that in this example the logout records do not name the associated account name that goes with the corresponding login records. You need to merge and sort the logs from both servers before you can reconstruct an accurate history of login/logout events.

Again, don't be afraid to get help from an expert.

How It Fits Together

When we conduct an investigation we collect bits and pieces of information from various sources. These sources vary in completeness and in reliability. The real point to this article is to talk about how to correlate the pieces together. When we do this we commonly run into several problems.

TIME-RELATED ISSUES

First, let's talk about the time-related issues. Most log files include some sort of timestamp with each record, which can be used to correlate entries from several logs against one another. One common problem we run into when correlating logs from different hosts together is that the clocks on those hosts may not be synchronized to the same time, let alone the correct time. You can sometimes infer this clock offset from the logs themselves. If the shell history file for my account on host A shows me running "telnet B" at time T1, but the TCP wrapper log on host B shows the Telnet connection at T2, then we can conclude that the clock offset between host A and host B is roughly T2-T1 (assuming they are in the same time zone). It isn't always possible to infer this offset directly, since there can be a significant lag between events in different logs (see below).

It is also important to know the time zone that each log was recorded in. Unfortunately, the timestamps in many logs do not include the time zone. Get into the habit of sending time-zone and clock-correction information when you send logs to others, and request the same when you ask others to send logs to you. I generally like to express time zones as offsets from GMT, since that is more universally understood and is less ambiguous than some of the common abbreviations.

Event lag is the difference in times between related events in different types of logs. For example, suppose that someone connects from host A to host B using Telnet and logs in. A Cisco Netflow log containing the traffic between A and B will record the time T that traffic to port TCP/23 (typically Telnet) on host B was first seen. If host B uses TCP wrappers to log access to the Telnet service, the log entries for that entry will probably have a timestamp very close to T. However, there can be a considerable delay between when a person is presented with a login prompt and when she actually completes the authentication process, which is when the `wtmp` record would be created. So I might see a NetFlow entry indicating attempts to connect to the Telnet service at 13:02:05, a TCP wrapper entry at 13:02:05, and a login entry at 13:02:38, 33 seconds later.

Event lag is important because often our only means of correlating entries from different logs together is through their timestamps. Unfortunately, since the amount of lag is often variable, we can't always correlate events specifically by starting time or even duration since the session in the network-traffic log would last longer than the login session. However, we can use session duration and starting time to eliminate false correlations – a login session that lasts 0:23:32 wouldn't (usually) match a phone session that lasts only 0:05:10. We can sometimes use the ending time of a session to make closer correlations, since the ending events often match up more closely in time. For example, logging out of a host you connected with telnet usually ends the Telnet session and its associated network traffic, so the logout event and the end of network traffic in the NetFlow log would be very close chronologically.

Sometimes logs are created in order of the ending time of a session, instead of the start time. This can lend further confusion to the correlation process. Log entries for Cisco Netflow logs are created when the "flow" of traffic ends. UNIX process accounting logs are created when the associated process ends. It is easy to misinterpret such logs, since important information may be buried much later in the log. Figure 3 shows the process accounting records corresponding to a login shell where someone ran `ls`, `cat`, and then a shell script that ran `egrep` and `awk`. Note that the `sh` processes corresponding to the login session and the shell script that was run show up after the processes started from within those shells. If you were just casually reading the log, however, you might miss this – I know I have on several occasions, and was very confused until I realized my mistake. Note that not all systems provide tools that print process accounting records in this format – the basic data is there in the file, but you might have to write some software to winkle it out!

```
                      start
line   account        time        duration     command
———    ————-          ———         ——————       ————-
ttyp1 romig           12:32:28    00:00:07       ls
ttyp1 romig           12:33:02    00:00:05       cat
ttyp1 romig           12:33:45    00:00:03       egrep
ttyp1 romig           12:33:45    00:00:04       awk
ttyp1 romig           12:33:45    00:00:04       sh
...
ttyp1 romig           12:20:12    00:10:02       sh
```

Figure 3: Process accounting records.

We can often can use the time bounds on one session to "focus in" on smaller portions of other logs. For example, if the modem-pool authentication records show a login session starting at 07:12:23 and lasting for 00:12:07, we can narrow our search through things like process accounting logs and other logs on target systems to just that time range (assuming that we've corrected for clock offsets and time zone). That's fairly straightforward, and we do this sort of bounding naturally. What may not be obvious is that we cannot always do this. Most of the log entries associated with a login session on a host should fall within the start and end times of that session. However, it is easy to leave a process running in the background so that it will persist after logout (using `nohup`), in which case its process accounting records will not be bounded by the login session.

MERGING LOGS

We sometimes have to merge logs made on different systems together to build a complete picture. For instance, on some occasions we have set up authentication servers that operate in parallel, in which case logout records may not be left on the same server that handled the corresponding login record. The Ohio State University now has two different routers that handle traffic to different parts of the Internet. There are some hosts where network traffic goes out through one router and returns through the second (due to asymmetric routing). If we are looking through Cisco Netflow logs for traffic, we now need to be careful to merge the logs together so that we have a more complete record of network activity. This can also be an issue in cases where we have multiple SMTP servers (records of some email will be here, some there) and for Web proxy servers.

RELIABILITY

Logs vary in the degree to which they can be relied upon to be accurate recordings of "what happened." Their reliability hinges on issues like the ownership and mode of the log files themselves. For instance, the `utmp` and `wtmp` logs on some UNIX systems are world-writable, meaning that anyone on the system could modify their contents. We are also dependent on the integrity of the system pieces that generate the logs. If those subsystems have been compromised or replaced, the logs that they generate may not be

Sometimes it isn't what we find in the log that is interesting, but what we don't find.

a complete or accurate portrayal. If an intruder has replaced the login binary with a "rootkit" version that doesn't record login entries for certain users, then the login logs will naturally be incomplete.

In other cases, the accuracy of the logs is subject to the security of the network protocols used for transporting the messages. Syslog and Cisco Netflow logs are both sent using UDP (the User Datagram Protocol), which makes no provisions to ensure that all data sent will be received. In these cases the logs can easily be incomplete, in the sense that records that were sent from the source were never received by the server that made the record that we are examining. This also means that it is relatively easy to create false log entries by directing carefully crafted UDP packets with spoofed source addresses to the log servers.

We can help guard against the dangers of incomplete or incorrect logs by correlating events from as many sources as possible. We will still have to adjust our theories to account for discrepancies among the logs, but at least these discrepancies will be more visible. This is especially true in the cases where system processes on a host have been modified or replaced by an intruder.

IP ADDRESS AND HOST NAME PROBLEMS

We need to realize that IP addresses can be spoofed and recognize cases where this is likely and cases where it is unlikely. (For example, spoofing is common in flooding attacks and rare for straight Telnet connections.) There is also a variety of games that people can play to steal domains, poison the caches on DNS servers, and otherwise inject false information into address/name lookups.

Unfortunately, many subsystems resolve the IP addresses that they "know" into names using DNS, and then only log the resolved names, which may not be correct. So we also need to recognize that the host names that we see in log files may not represent the correct source of the traffic that generated the log message. It's generally best for log messages to include both the IP address and the name that it was resolved to, rather than one or the other. If I had to choose one, I would choose the IP address, since that's more correct in most contexts (in the sense that the subsystem "knows" that it saw traffic with a source IP address of A.B.C.D, and we can't know whether the resolved host name for that is correct).

RECOGNIZE WHAT'S MISSING

Sometimes it isn't what we find in the log that is interesting, but what we don't find. If we see NetFlow data showing a long-lasting Telnet session to a host but no corresponding login entry for that time period, this should naturally raise the suspicion that the login entries are incomplete (or that the NetFlow data was incorrect). If a shell history file shows that someone unpacked a tar archive in /dev/ – but we cannot find /dev/ on the system – then someone has either deleted it or it is being hidden by a rootkit of some sort.

Some Comments on Specific Logs

I have a few parting comments about some of the logs that we commonly work with, in light of the issues that I've addressed in this article.

PHONE LOGS

I don't know whether the phone companies do anything to synchronize the clocks used for timestamping phone trace logs; past experience shows that they are usually close to correct, but are usually off by a minute or two. Note also that there can be significant event lag between the start of a phone connection and the start of an authenticated session on the modem pool that someone is connecting to (or start of activity in other logs). The easiest way to match calls to login sessions and other logs is by narrowing down the search by very rough time constraints and especially by call duration. We tend to have many short dialup sessions and relatively few long sessions, and so it is generally easier for us to match login sessions against longer phone calls, since they are "more unique" than the shorter calls. For example, there are few calls that last at least 2:31:07, but many that last at least 00:05:21.

Note that the most we can say in these cases is that the process-accounting records are consistent with running the script "blub."

UTMP, UTMPX, WTMP, AND WTMPX LOGS

Apart from the reliability concerns mentioned above, on some UNIX systems you also run into problems that are due to the fact that the wtmp and utmp files truncate the source host name (for remote login sessions) to some limited size. This obscures the source host name if it is long. One way to help address this is to use other sources (like TCP wrapper or network traffic logs) to try to determine the correct host name.

UNIX PROCESS-ACCOUNTING RECORDS

One problem with process accounting records is that they only contain the (possibly truncated) name of the binary that was executed, and not the full pathname to the file. Consequently, to find the binary that belongs to a process accounting record, we need to search all attached filesystems for executable files with the same name. If there is more than one file, it may not be possible to specifically determine which binary was executed. In the case of shell scripts, the name of the interpreter for the script is recorded (e.g., Perl, sh, ksh), but the name of the script isn't recorded at all.

In some cases we can infer the name of the executable on the basis of other records, such as shell history files and by examining the user's PATH environment-variable settings. If we see from a user's shell history file that a command named "blub" was run at a given time, and a search of attached filesystems reveals a shell script named "blub" in a directory that lies in their "PATH," we can reasonably correlate the file with the shell history file entry and the process accounting record for the shell that was invoked to interpret the contents of "blub." We should be able to make further correlations between the contents of the script "blub" and the process accounting record if the script executes other programs on the system. This is especially true if the sequence of commands executed is unique, or the commands are not commonly used in other places. Note that the most we can say in these cases is that the process accounting records are consistent with running the script "blub." We cannot prove directly from the process accounting records that the script was what generated those log entries – for instance, a different script named "blub" might have been run, and then deleted or renamed.

UNIX SHELL HISTORY FILES

Some UNIX shell history files are timestamped – otherwise, it can be very difficult to match these records to other events, such as process accounting records. Note, of course, that shell history files are typically owned by the account whose activity they record, and so are subject to editing and erasure. You should be able to match the events depicted in the shell history file against the process accounting records and sometimes against others, like logs of network traffic, timestamps on files in the local filesystem, and so on. The shell history is written when each shell exits, so overlapping shells can obfuscate the record. (History is written by the last to exit. . . .)

There's a wealth of information available in other logs on a system, especially if the log levels have been tweaked up by a knowledgeable administrator.

SYSLOG, NT EVENT LOGS, AND OTHER TIMESTAMPED LOGS

There's a wealth of information available in other logs on a system, especially if the log levels have been tweaked up by a knowledgeable administrator. Take note of my cautions above about correlating log entries by timestamps and about the reliability of the logs. It is ideal if you can log to a secure logging host so that an intruder can't easily modify previously logged events. This is easy to do with syslog, and fairly easy to do with NT event logs using both commercial and free software. There's even software that allows you to "transcribe" NT event-log entries to a syslog server. One thing to beware of – with syslog, the timestamp that appears on the entries in the log file is the time that the entry was received by the local machine according to its own clock, not the clock of the machine that the log entries come from. That's generally a good thing, since you've hopefully taken pains to synchronize your syslog host's clock to "real time." However, it can cause confusion if you try to correlate those log entries to other events from the original host, since there may be a clock offset between that host and the syslog host.

OTHER SOURCES THAT WE HAVEN'T TALKED ABOUT

There's a wealth of information that can potentially be found on the local host – binaries, source code, output from commands run, temporary files, tar archives, contents of memory of various processes, access and modification times for files and directories, files recovered from the free and slack space on the filesystems, information about active processes, network connections and remote filesystem mounts at the time of the incident, etc. You need to hunt for these and fit them into your reconstruction of the history of the event. For most of this information, unless you have access to more detailed logs (e.g., timestamped shell history files or tcpdump captures of the Telnet session where the intruder did his work), a lot of this reconstruction will necessarily be informed guesswork. Suppose we find a process running on a UNIX host and run lsof on it. (lsof lists the file handles that a process has open – very handy for investigations where processes have been left running.) If lsof reveals that this process has open network connections, we might be able to correlate these against entries from network traffic logs based on the time, the host's IP address, the remote IP address, the IP protocol type, and the UDP or TCP port numbers (if applicable).

Take-Home Lessons

There are a few practices you can follow to improve the condition of your logs and make it easier to correlate them against one another. First, turn your logging on and log a reasonable amount of data (both in quality and in quantity). Disks are cheap these days, so you can afford to both log more and retain it longer. It is always a good idea to forward copies of your logs to a secure log server – this is easy to do with both syslog and NT event logs. Synchronize your clocks to a common source – if you don't want to synchronize them to an external source, you can at least set up a fake internal source and synchronize them using the network time protocol. If you have a choice, log IP addresses in addition to (or instead of) the host name that corresponds to the address – the host name might be more meaningful to you, but the IP address is more correct. Finally, secure your systems so that you don't have to do these sorts of investigations often!

nessus: the free network security scanner

by Renaud Deraison

Renaud Deraison was tired of people complaining of the cost needed to bring their network to a decent level of security, so he started to write free tools to help them to achieve their goal at a much lower cost.

<deraison@nessus.com>

and Jordan Hrycaj

Jordan Hrycaj works as independent security consultant and joined the Nessus project in late 1998. He believes that clever system solutions are always born in the mind rather than designed with the latest development tool.

<jordan@nessus.com>

RENAUD DERAISON AND JORDAN HRYCAJ ARE THE AUTHORS OF NESSUS AND THE FOUNDERS OF THE NESSUS CONSULTING S.A.R.L.

A network scanner is a tool for analyzing network services, available on a given set of systems. With Nessus, a new breed of scanners has been published capable of running real attacks, often called exploits, in order to determine that well-known system deficiencies can be exploited when running the attack against the scanned systems.

History

When Nessus was born back in 1998, it was just cool to have a free network scanning and attacking tool with design goals similar to SATAN, written by Wietse Venema and Dan Farmer. Right from the start, Nessus was set up as a client-server tool endowed with its own communication protocol. The scanning and attacking workload was put onto the server, and the presentation of the data was done by the client, very similar to the design of SATAN.

In addition to that, the client realized better online control. So each host under scan and attack could be released from the scanning, individually, at any time. SATAN's design launched the server and waited for the scanning to complete, without any control over the process. The attacks used by Nessus only test for vulnerabilities and do not actually perform a "break-in."

Nessus was planned and introduced to be publicly supported as a free software project. Seen from an organizational standpoint, this only meant that the source code of both the client-server platform and the plugin code database (the implementation of the attacks and the scans) are open for public use and discussion.

Licensing Concept and Support Considerations

Nessus has been released under the GNU Library General Public License (renamed to Lesser GPL in 1999), which might be further restricted, partly by some contributions to Nessus.

Within one tool, a freely available set of working proof-of-concept attacks has been published. This is still unique, as the size of the Nessus database is far beyond that of any other scanner, even commercial collections.

The authors of Nessus strongly believe in the free and open-source approach. This has a clear impact on the general acceptance of and contributions to Nessus. Many bugs and exploits are probably found by individuals, favoring a public and open audience rather than making a quick buck with a company that solely handles the exploits as classified information.

The software can be deployed, tested, and modified freely. There is public bug-track management and a searchable mailing list. Additionally, professional software support is offered for commercial users to provide (legal) support contracts.

Implementation Notes

With the scanning and attacking database, Nessus aims to be as complete as possible. It currently performs over 500 security checks. This includes advanced Windows NT checks such as testing for permission to access the registry keys remotely, or for inappropriately shared partitions.

Nessus has been designed to be easily installed and handled by a user or an operator.

While attacking, the intention is not to miss any vulnerabilities whatsoever. For instance, nobody prevents you from opening a Telnet service on port 32 rather than 23, and a testing tool should be able to find that out. Nessus will actually probe open ports with unusual port addresses to see if Telnet or something like it is running there. Being that flexible has not been common for a long time and probably is still uncommon, especially with commercial software.

Nessus does not guess a host or operating-system type by reading the greeting message banner of the Telnet program. Long after QUESO and NMAP introduced the IP-stack fingerprinting approach, the banner method is still common practice with many other tools.

A Strategic Tool

As of today, Nessus has been used as a tool to enforce the security policy of a company site, institution, or organizational entity. Nessus goes much further than answering questions like "Does my firewall have the particular bug reported in the BugTraq list the other day?"

The Nessus project aims to provide a tool to check out and analyze the network as seen from a security standpoint that is

- comprehensive and reliable
- distributed
- continuously up-to-date
- well known
- cost effective

In the strategic setting up and running it has some similarities with network probes commonly installed and used to monitor data and voice traffic in quality and quantity.

Although the resulting reports are not always simple to grasp by nature, Nessus has been designed to be easily installed and handled by a user or an operator. It is possible to control a session in batch mode as well as with a full operator dialog. The server poses access restrictions upon the controlling operator using public-key technology. Once installed, the operator can have full and individual control over a farm of servers, possibly without the need to remember passwords (of course, the workstation needs physical access security, unless the keys are protected by a pass phrase).

With the arrival of public bug-registration sites like CVE, Nessus easily integrates and contributes to the worldwide network of security-relevant information systems that are freely available for everybody.

Architecture

CLIENT-SERVER COMPUTING

The server, named nessusd, is the smart part of the program, which is in charge of the security assessment, and is available for modern POSIX-like systems such as Linux, FreeBSD, OpenBSD, and Solaris. There might be more but they are not officially supported by the core team. The client, as supported by the same team, is additionally available for the Microsoft Windows releases 9x, NT4, and W2K.

The client is the controlling front end to the server. The communication between the server and the client is encrypted. Session negotiation and authentication on the server is based on public-key encryption technology.

The nessusd server manages its own user and access database, so different scan and attack privileges can be configured. It is, for example, possible to configure the nessusd server so that each user can test only her or his own computer.

PLUGINS

The nessusd server is an application platform for running a series of network-based test programs and attacks, the results of which are collected in a common database. These programs, called plugins, have access to this database. Apart from storing results, they also use it for communication and optimizing tests.

In a few cases, plugins are dynamically linked program fragments (usually called shared objects, or shared libraries.) Most commonly, though, they will be interpreter scripts in a language, called NASL (the Nessus Attack Scripting Language). These scripts can be run immediately and independently of any operating system by nessusd.

The NASL interpreter handles the communication between the scripts transparently through the database, mentioned above. The script language is limited in its power to implement applications different from network tests and attacks. It is not designed to run in a sandbox as TAINTPERL and Java do, but does control what actions can be carried out through the design of the interpreter.

Thanks to this architecture, updating a set of security checks for nessusd is usually just a matter of downloading some files and copying them to the appropriate place on disk. And this task is automated by shell scripts like `nessus-update-plugins`, which retrieves all the newest NASL scripts, installs them at the proper location, and reloads them into the nessusd server. The latest NASL scripts available are regularly published on the Nessus script page.

Deployment Topology and Interfaces

Currently, Nessus supports only the deployment of standalone nessusd servers with multisession support. Secure server-to-server communication for distributed attacks is possible but so far has been implemented at transport level only.

There are well-defined library APIs for the NASL interpreter and the PEKS-encrypted communication channel API. There is also a well-defined text form used for storing the scanning and attacking results. A database API has been under discussion for some time.

Availability Notes

The whole Nessus Package is about 16MB in source code; extra library packages needed, like `gmp` or `pcap` add about 4MB. The exploits-and-attack database is currently somewhat larger than 2MB of source code. Altogether, the gzipped sources make up a bit more than 3MB.

There is a network of worldwide FTP mirrors; the easiest way to access them is to browse any of the Nessus Web sites (<*http://www.nessus.org*> being the primary one). On these sites, some online installation instructions are also available, as well as the screen shots of a sample session.

Although version 1.0 was released not so long ago, Nessus is under active development. The next major release will have better handling of large networks (over 10,000 hosts), will offer the ability to do distributed scans, and will have better multilingual support. (Currently, most plugins have English and French descriptions and messages.)

Summary

Nessus is a free network-security scanner and attack tool with a clear strategic focus. Its main goal is to help enforce the security policy of the network site that is tested and attacked. Designed as a server-client system, many servers can play the role of monitoring devices controlled by one or more client operators.

Nessus is not a one-shot or standalone tool. It can be used that way, but is designed with clear interfaces and APIs. This allows further development and integration at a public or individual level.

Nessus has been developed in Europe, so there are currently no export restrictions whatsoever.

security devices that might not be

And How to Approach Them as a Consumer

by Mudge

Mudge is Vice President of Research amd Development for @stake Inc.

<mudge@atstake.com>

Many times we, as consumers of products for the online world, make assumptions about those products' security stance. Everyone would love to assume that any commercial piece of software that they purchase is "secure." After all, it says so on the box. This is a common problem. What about the devices that have an implied security connotation when in fact they might not? Conversely, what about devices that appear to have no bearing on security but upon closer inspection are critical to an infrastructure?

While engaged in some network-design work in the @stake labs, my team and I came across crypto-accelerator appliances. The one in particular that we examined at the time was a self-contained unit. It would boot and run from a memory card and take the burden of encryption off of the end node. In other words, it would act as an invisible device (like a hub) and take HTTPS streams in from the outside world and output HTTP streams on the inside. From the inside nets to the external networks the device would take the HTTP streams and output HTTPS for the appropriate session. Thus the device was required to keep state and session information locally.

Here is an example of a device that contains a public key and a private key, presents a credential as if it were the final end node, and is conducting cryptographic transforms on data passing through it. Instantly one is led to the conclusion that this is a security device. However, closer examination will show that this is not the case and might even present liabilities.

A crypto-accelerator of this type is designed to offload computational work that is processor-expensive for systems. Oftentimes this is done through dedicated hardware on the appliance in custom ASICs. This reduces the load on the end system general-purpose processor so it can go back to serving content, accepting credit cards, and kicking out instructions to other systems as to where to send the goods. Yes, it is in fact a load balancer or coprocessor in nature, much like older systems where you could opt to have a math coprocessor. Few people would think of a math coprocessor as a security device; instead most would consider it a load balancer of some ilk where it is taking the expensive operations and handling them for the main CPU. In reality, though, it could very well be performing the math portions of cryptographic transforms. Here, the device is removing the security blanket to speed the processing on the data within.

Simply having the words cryptography, crypto, crypto-accelerator, certificates, SSL, HTTPS, etc. in a product name or description gives the consumer the impression that what is being used is a security device that is putting security into the mix – not removing it. This is not necessarily the case.

The appliance here is not intended to protect the end systems. It is not even claiming to protect itself. In fact, one can argue that it is now more important to secure the back-end network, as the traffic is not actually encrypted all the way to a final destination, and thus the potential for monitoring and compromise of confidentiality is exaggerated.

If you see a device in your network that is designed to be appliance-like and offer security, be very suspicious.

Does this present a problem? Only if it caught the consumer off guard. A little analysis up front can go a long way.

- The device is used to remove a security layer.
- The device is designed to be largely “plug and play.”
- The device is an embedded system with no moving parts.
- The vendor offers remote support.
- The owner can remotely manage the device.
- The owner can locally manage the device.

If we abstract the above to more generalized security devices, or nonsecurity devices that have an implied security component, we can take the first four items above and elaborate a bit.

THE DEVICE IS USED TO REMOVE A SECURITY LAYER

In the real world this unfortunately often translates to a lax security stance in the design stage. The goal in the above example is to strip the HTTPS coming in on one end and spit out raw HTTP on the other. A relatively simple goal, if that is all one is thinking about. If one were working in the other direction, of introducing security in an embedded system, one would (hopefully) think about how to harden the system itself. The notion of not caring about the identity of the end node connecting, just that the session is encrypted but not necessarily authenticated, lends itself to this poor stance. This is an important area to analyze before deployment. Was the vendor lackadaisical and not treating the device as security relevant?

THE DEVICE IS DESIGNED TO BE LARGELY “PLUG AND PLAY”

This should almost always raise a large, red warning flag when seen in conjunction with “security devices.” If there were a silver bullet, one-size-fits-all solution, then there would be no need for all of the different products and vendors. There would be one operating system. No need for public markets, etc., etc.

To be honest, Microsoft even gets a somewhat unfair rap on this count for security. One of their main goals is to sell an operating system that is ubiquitous. To do so their product must need minimal – or more appropriately no – custom configuration in order to work in all environments. The same build-and-stock configuration must exist in academic, military, corporate, medical, and personal environments. A custom build for each area and the associated support costs would be prohibitive. We wonder why there are so many security ramifications? Because we, the consumer, have demanded that it be largely “plug and play” for all environments. If you see a device in your network that is designed to be appliance-like and offer security, be very suspicious.

THE DEVICE IS AN EMBEDDED SYSTEM WITH NO MOVING PARTS

So what if the component in question is a more or less dedicated system? Chalk one up toward a step in the right direction. In many cases it is much easier to batten down the hatches on a product or system that is designed to do one thing in one particular environment, and that alone. There very well might not be all of the problems associated with a generic one-size-fits-all system. Then again, there is also the strong possibility that the embedded system was chosen simply for cost and in reality is just a generic system on the inside. Even if it is not a generic OS, did the vendor really take security seriously, or are there tell-tale signs that point to less than master-craftsman type work?

Here are a few of the things we have seen in "embedded" appliance devices:

- entire generic OS running on flash memory cards – not secured in the least
- poorly crafted and tested TCP/IP stacks on ASICs
- proprietary chips without tamper-resistant epoxy on them
- serial EEPROMs with programming leads exposed
- tamper-evident tape placed on the inside of the appliance where it is not visible

THE VENDOR OFFERS REMOTE SUPPORT

If you are lucky, the vendor knows one of the passwords of an account that you set up for him. More often, the vendor is aware of a hidden account that you were not told existed. While this is arguable, even if it is done for truly nonsecurity-related devices (what are those?), it should be a career-limiting move for the marketing or sales person that originally decided this was required to sell a security device. Does this still happen? Unfortunately so – the crypto-accelerator mentioned above contained a couple. We have also found them in printers, hubs, and plenty of software servers and clients. Of course, the remote support might be something more obvious such as a modem and analog line, or perhaps it was given away when customers asked for yet more holes to be placed in the firewall to allow them to get in for troubleshooting and diagnostic purposes.

Does this happen on your network? How strong is the stack on that VPN box? Let us rephrase – how strong is the stack on that VPN box that you deployed parallel to the firewall? Are the infrastructure components such as switches and load-balancers managed in-band or out-of-band? How many addressable devices are on your network and how many of them were able to be dropped on the network right out of the box and they basically configured themselves? Does that NTP server offer more than just the correct time? Are your hubs and switches addressable? Why?

Hopefully this article has caused some to think about their current environment and others to take a different look at the items they are about to deploy.

Sleep well.

[Editor's note: Peter Guttman's paper, An Open-Source Cryptographic Coprocessor, <http://www.usenix.org/publications/library/proceedings/sec2000/gutmann.html>, *makes an excellent companion to this article, with very concrete examples.]*

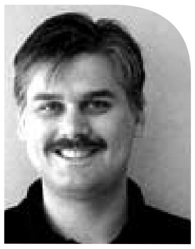
scalpel, gauze, and decompilers

Dissecting Denial of Service (DDos)

by Sven Dietrich

Sven Dietrich is a senior security architect for Raytheon ITSS at the NASA Goddard Space Flight Center. His focus is computer security, intrusion detection, the building of a PKI for NASA, and the security of IP communications in space.

<spock@netsec.gsfc.nasa.gov>



You walk into your office on a Monday morning and find that your usual game of Quake is painfully slow, like molasses in deep winter, and that your email inbox is scarily full with tons of messages from stricken users complaining about the network. Phrases like "the network is slow," "DNS is not working," and "I can't get to my Web site" ring in your ears as you retrieve your voicemail. Chances are, you are under a DoS (Denial of Service) attack.

What should you do? The most important goal is to determine whether you are in a simple DoS attack or a more complex DDoS (Distributed Denial of Service) attack, a distributed variant. The latter attacks began occurring on a large scale during the summer of 1999. Groups of intruders using massively automated compromise tools began infiltrating a large number of computer systems worldwide in late July and early August of that year. For what purpose, you may ask. The first sign of malignant behavior I encountered was in the form of a script that first mentioned a name that would go around the world, literally: trin00 a.k.a. Trinoo.[1] The script itself was copying in a program called leaf and placing it in a typically unsuspicious location on a UNIX system: /usr/bin/rpc.listen. For the inexperienced, this may appear to be a legitimate process taking care of RPC (Remote Procedure Call) services.

Well equipped with rootkits, the intruders would even hide that process on occasion and most likely would have gone unnoticed, except for one thing: the load level. Due to a flaw in the program, a cron job was launched every minute to keep the program, which came to be known as a DDoS agent, alive. The name leaf seemed to imply a tree, a leaf node of a larger structure.

The Scalpel

So what was the big deal? We all have seen DoS programs, whether they are UDP flooders, ICMP flooders, spider programs, Smurfers, or even the original Morris worm of 1988. What made this one different became apparent when I took out my electronic scalpel and started dissecting this piece of code: references to a "master server," IP numbers, and traces of crypted strings and commands. The electronic scalpel, of course, is your favorite binary or "hex" editor, but the UNIX strings command will also do in a pinch. Only a few weeks later, the next attack wave unleashed a packet storm known as the "University of Minnesota incident," in which roughly 250 systems bombarded the networks of the University of Minnesota and brought them to a screeching halt for three days.

What was going on? The packets were coming in so fast, they must have come seemingly from everywhere, at very high rates. The packets were very short, only 40 bytes in most cases, but the sheer rate at which they were aggregating at the target, the University of Minnesota, was simply overwhelming. So what was it really? Digging further into the code and tracing the packets to their apparent source led to the discovery of the "master server," a.k.a. the DDoS handler. This was the program "coordinating" the attack, ordering the DDoS agents to flood the target with packets. The intruders, counting on being discovered sooner or later, had taken precautions: The list of all the DDoS agents was encrypted, and in some cases the encrypted file was unlinked. Extreme precaution was needed to recover the full list of agents. As agents were discovered and shut down, new ones were being added in order to maintain the constant flood of packets. It was indeed a large attack structure. In a better-covered set of events, the incidents of February 2000 involved several well-known e-commerce and industry sites as targets of DDoS attacks.[2]

So we return to the original question: What should you do? First and foremost: Start recording packets. Network (and computer) forensics have been and still are the key instruments in tracing back to the attacker. A simple tcpdump[3] process at your site border(s) will do for lack of a better choice. Then, and only then, start talking to your upstream Internet service provider. With high probability, the source IPs of the flood packets are spoofed, making determination of the actual source a bit harder at first. A good relationship with that provider is critical to getting a fast response, as floods can last anywhere from a few minutes to hours or sometimes days. Let us keep in mind that several scenarios are possible:

1. You are the victim.
2. You are the host for one or more DDoS agents.
3. You are the host for one or more DDoS agents and a handler.

What should you do? First and foremost: Start recording packets.

The Gauze

Of course, variants and combinations of the above are indeed possible. In the case of 1, most likely you would like to restore your connectivity as soon as possible. Talking to your upstream providers should help locate the source of the floods, as they may have a better view of the flows of the (spoofed) packets. Of course, it is beneficial to get each intermediate provider to start recording packets for later inspection. Once one source is located, case 2 applies, as follows.

Now is the time for a tricky decision. Two types of traffic are involved in a DDoS situation, the flood traffic – felt heavily at the target, not necessarily at the source – and the very light control traffic. The control traffic is the interesting one, as it will be the path to the DDoS handler if no references to it can be found in the DDoS agent code. So what should one do? Shut down the DDoS agent system cold and subsequently make a binary snapshot, e.g., using dd, of the hard disk by transporting it to a different physical system? Maybe. Or should you take a snapshot while it is running? By experience it has been a good idea to "freeze" the system by doing the former.

There is really no simple answer, as it may be necessary to preserve the contents of memory also. Using lsof[4] and forensics tools such as The Coroner's Toolkit[5] are definitely a good idea, as things may not be as they appear. Of course, you are still recording packets at this point and any attempt at contacting the agent will be recorded in the logs, hopefully. The important aspect is to seep up as much evidence as possible with our electronic gauze, for later scrutiny. In any case, seek the advice of a DDoS foe, a CERT or FIRST member.

In some cases, law enforcement may choose to have its own idea of what to do in that situation and can advise you. That is outside of my domain and I shall not comment. For general guidelines, see the original CERT report[6], compiled by a small group of experts.

In the case of 3, you win! You are the lucky host of a DDoS handler and possibly of a few DDoS agents. Now it is doubly important to proceed with extreme caution in order not to destroy any evidence that may lead to the discovery of the entire list of agents, the handler and/or the agent code, the possible source code, and last but not least, the path to the actual intruder.

The number of vulnerable systems is increasing exponentially.

So, you think, why not just remove the handler, reformat the drive of the computer, reinstall a more secure version of the operating system, and forget that it ever happened? Well, it is complicated. The intruders have considered that possibility and taken precautions for one or more "backup handlers" to take over the existing agents and continue their flooding. Think of it as a bad weed: Unless you get to the root of it, it will come back to haunt you. Taking a closer look at your network will help identify other, potentially dormant, agents. For scanning your network for well-known agents or handlers, see [7, 1, 2,] . Similarly, one can scan the host for well-known agents or handlers. Unfortunately, these programs are not always "well-known," and host integrity checking is an important asset in determining what files, if any, have been added or modified.

Of course, I have experienced quite a variety of "mishaps" during my quest for DDoS tools – from system administrators not wanting to surrender their tapes, not performing the correct backups, not responding altogether, or not acknowledging that their systems were compromised, to just reinstalling the operating system on the compromised host without prior backup. The good times, however, are finding the actual code for the tools, mostly in binary form.

Once one has the actual tool, how should one proceed? While it is often possible to replicate the traffic and provide substantial guidance to intrusion-detection system configurators, research prototypes and commercial variants, it is far more exciting to find the real code, the source, in order to discover potential flaws in the algorithm. These flaws are the mechanisms for registering flood or control traffic that might otherwise go unnoticed or dismissed as "noise." In the Shaft case[7] analysis of the agent source code revealed a fixed TCP sequence number for flooding. The "echoes" of attacks have been felt elsewhere in the world and are now signs of an ongoing Shaft attack. Yet other flaws may reveal mechanisms for shutting down floods in progress, such as improper authentication of handler-to-agent communications.

The Decompiler

In a different setting, reverse engineering of the binary executable may be the last resort at getting a deeper understanding, short of reading straight COFF or ELF executable binaries. On to the next ace up our sleeve: decompilation. In the case of Mstream[8], hand decompilation was essential to obtaining the attack algorithm, as no source code was immediately available, and informing the appropriate audience of our findings. So far, the decompilation process has been very much human, as the decompilation problem is hard and remains more of an art than a science.

Conclusion

As we lie in the aftermath of recent advertised and not-so-advertised DDoS attacks around the globe, we find ourselves still faced with the threat of ever-evolving DDoS tools. [9, 10] We need rapid incident-response teams, cooperative ISPs, good network forensics, good host forensics, well-established policies, and competent and DDoS-aware experts, all wrapped into one. While there are some good starting points for limiting the impact of such attacks, or even using traceback for identifying the source of the packet floods, [9, 10] that is outside of the scope of this article; the reader is referred to [2, 7] as starting points.

Nevertheless, the number of vulnerable systems is increasing exponentially, providing more hunting grounds for what apparently started as an IRC-channel takeover war. The fact remains: DDoS attacks should be our concern, as they represent a powerful tool to disable or incapacitate government, e-commerce, industry, and educational sites alike, in some cases even the underlying infrastructure.[9] Thus far, intruders launching those attacks have successfully evaded detection through their cleverly built DDoS networks and are continuing to taunt us. A coordinated response and ongoing research will hopefully put us one step, or at least half a step, ahead.

REFERENCES

1. David Dittrich. The Trinoo Distributed Denial of Service Attack Tool. 21 October 1999.

2. Sven Dietrich's DDoS page. <*http://netsec.gsfc.nasa.gov/~spock/ddos.html*>.

3. <*ftp://ftp.ee.lbl.gov/tcpdump.tar.Z*>

4. <*http://vic.cc.purdue.edu/*>

5. Dan Farmer and Wietse Venema. The Coroner's Toolkit. <*http://www.porcupine.org/forensics/tct.html*>

6. CERT. Results of the Distributed Systems Intruder Tools Workshop. December 1999. <*http://www.cert.org/reports/dsit_workshop.pdf*>.

7. Sven Dietrich, Neil Long, and David Dittrich. Analyzing Distributed Denial of Service Tools: The Shaft Case. In Proceedings of USENIX LISA 2000, to appear.

8. David Dittrich, George Weaver, Sven Dietrich, and Neil Long. The "mstream" Distributed Denial of Service Attack Tool. May 2000. <*http://staff.washington.edu/dittrich/misc/mstream.analysis.txt*>.

9. Sven Dietrich, Neil Long, and David Dittrich. The History and Future of Distributed Systems Attack Methods. 5-minute presentation at IEEE Symposium on Security and Privacy, Oakland, CA. 16 May 2000.

10. Sven Dietrich. Dietrich's Discourse on Shaft (DDoS). Work-in-Progress presentation at USENIX Security Symposium 2000, Denver, CO. 17 August 2000.

an interview with Blaine Burnham

by Carole Fennelly

Carole Fennelly is a partner is Wizard's Keys Corp, a company specializing in computer-security consulting. Carole also writes for www.sunworld.com.

<fennelly@wkeys.com>

Dr. Blaine Burnham is Director of the Georgia Tech Information Security Center.

We have all heard the design model "Keep It Simple, Stupid" (KISS). In his keynote address at the USENIX Security Conference in August, Dr. Blaine Burnham expanded on this concept of common-sense security architecture by demonstrating his points using examples that everyone could easily identify with.

I found many of Dr. Burnham's points to be quite clear and inarguable. In discussing the principle of Acceptability, he stressed that a security solution that is too difficult to use will invite people to go around it or not use it at all. I couldn't agree more.

Some of Dr. Burnham's statements were thought-provoking and invited further discussion. He graciously agreed to take time out from his busy schedule to answer a few questions.

Design Principles of Simplicity: Followup Questions

Carole Fennelly: There was a reference to code that is not open source as providing security by obscurity. While relying on obscurity as the sole means of providing security is foolhardy, isn't some obscurity necessary? There was a comment later in the talk that "it takes a secret to keep a secret." Isn't this a form of obscurity? Isn't privacy also a form of "security through obscurity"?

Blaine Burnham: "Security by obscurity" speaks to the notion that you are basing the security of the system on the assumption the bad guys are unable to discover the internal working of the security system. Historically this has been a very bad assumption. We always tend to underestimate the ability and persistence of the bad guy. This is not to say that one should aggressively market one's security architecture to the bad guy. The only safe assumption is to assume the bad guy has a complete and accurate copy of your security solution.

Regarding the "it takes a secret to keep a secret" statement: It simply means that the solution is designed in such a fashion that the introduction of secret content enables the system to propagate the ability to keep a secret. There is nothing obscure about the secret – usually everything about the secret – except its actual content is known. For example, the DES algorithm is widely available. The details of generating DES keys are openly available. However, a secret (a specific instance of a key known to only one party) DES key can reliably protect – keep secret – a great deal of information.

I don't see privacy as a form of security through obscurity. To me privacy is a global system property/behavior in which the system has access to the private information but it does not divulge the information in violation of the privacy policy. The system knows it doesn't tell. Part of the problem has been the absence of meaningful privacy policies – hence an open season on personal/private information, a behavior that argues that personal information is the property of the holder, not the referent – and therefore the referent has no control/stake in the information. In addition, we have to deal with the fundamental weakness of the systems to enforce any meaningful privacy policy in the face of anything more than casual attempts to assault the system.

Carole: Actually, what I was referring to with regard to privacy fits in with your explanation of "security by obscurity." I may not aggressively advertise where I live and my bank account numbers to the public at large, but I don't rely on that "obscurity" to protect myself.

Blaine: This is a good working example of my point. You don't have to advertise and otherwise aid and abet the bad guy. On the other hand, these measures in and of themselves cannot provide you the real protection you may need. Some mechanism(s), usually of a completely different nature, will have to be employed to provide the protection you may demand.

Carole: A comment was made that script kiddies create so much "noise" that it is difficult to track the real criminals. Isn't some of this relatively harmless noise necessary to raise awareness of security in the corporate world?

Blaine: I would not like to argue that this noise is harmless. In fact it is very harmful – depending on who you read – latest numbers put the cost in the trillions. Further, as distressing as it is, the observation that the security awareness of the corporate world has been significantly increased as a result of this noise appears to be true, at least to a first approximation. I find this whole "motivational" discussion tremendously upsetting because it shouldn't have to happen. There has been any amount of discussion and ample demonstration, for years, pointing to the encroaching risk to information systems. I find it unbelievable that we have done so little, really, to address the problems. I suspect that something like a consumer-protection agency is going to come about to deal with the problem. This will be a solution that no one will like.

Carole: I certainly don't endorse the activities of script kiddies and I agree they are a major annoyance. But aren't many reports of "damages" grossly exaggerated? Such as reporting the damages as including the cost of installing a firewall and redesigning a Web site?

Blaine: I haven't spent much time trying to validate the legitimacy of the damage claims. I know the impact of any of these DDoS attacks can be very substantial.

Carole: You mentioned that insurance companies will become an incentive for improving security. Do you think they will have a different picture of actual damages? Won't they hold organizations liable for not adhering to industry best practices?

Blaine: I think the insurance industry will have consistent measures for assessing the damage. What those measures are has yet to be determined. But over time insurance firms have demonstrated the ability to home in on the correct measures. I don't exactly see how the insurance industry will hold organizations liable. I think it will work more along the lines that failure to adhere to best practices may void a company's insurance policy. Sort of like – as I recall – skydiving can void a personal injury/life insurance policy. However, in addition, the interdependencies of e-mumble will create situations such that if a particular business fails to adhere to best practices and the consequent damage propagates to the e-mumble business partners, the insurance representatives of the damaged parties will come at the nonadhering business for compensation. This could have *enormous* consequences. For example, if you are running some mom-and-pop telecommuting engineering function for a major toy company and you are networked into their whole just-in-time manufacturing – for the Christmas rush – toy production facility, and you don't take sufficient protection measures while you are sitting on the beach somewhere while you put the finishing touches on your design, and the bad guy (today he may be in the employ of a competing toy company, tomorrow who knows) is able to gain access to your system and alter the design you upload to the JIT plant, and the plant manufactures the toy with a lead-based paint (this is the bad guy's modification) that causes the toys to all be recalled the day after Thanksgiving. I would hope you had paid up liability coverage – *a lot of it*.

Carole: You stated that "hostile and malicious code are the real problems." What about badly written code?

Probably one of the more significant overlooked notions is the word "personal" in the phrase "personal computer."

Blaine: The Greeks built the Trojan horse after spending tremendous energy exploring for a more direct access to the city of Troy. It is fair to observe that the Trojans were probably fairly disciplined in their walls and gates and windows maintenance. Had they not been, the Trojan horse would not have been necessary. Look at it from the bad guy's point of view: Take advantage of the target's mistakes; these mistakes lower the cost of the effort to achieve the objective. Badly written code is a tremendous advantage to the bad guy. He doesn't have to work so hard.

Carole: You stated that "security is not an add-on." How do we enforce this? If you look at the white paper for the proposed Simple Object Access Protocol (SOAP), security is certainly considered to be someone else's problem.

Blaine: I cannot argue for or against better alternatives for the SOAP; however, at least the SOAP does not claim to support security services. There is no confusion about this. Don't look to SOAP for security services. Q.E.D.

Carole: How can we make security attractive to the "bottom line"?

Blaine: This has been tough. I have tried to picture/market security as a business enabler. This sometimes works – sort of. I think the issue of "due care" will eventually work its way into the auditing and insurance side of the business and businesses will have to respond. I don't see this approach delivering the technology we really need for the information age that is upon us.

Carole: There was a reference to home schooling using the Internet. While the Internet can be a great source of information to children, isn't physical socialization also important?

Blaine: Probably, but I think it will be *way oversold* by the folks that Internet-enabled home schools will threaten the most. For the most part children today can have/get as much "socialization" as they can schedule/stand, outside of the conventional school environment. Internet-enabled home schooling will allow families to choose the socialization they want, rather than have to deal with the "socialization" being forced upon them. For a lot of reasons we have let our schools degenerate into war zones in which bullies reign. Additionally, many, many parents feel the schools have abandoned any notion of a wholesome, family-centered system of values. For them and for many others, particularly families with talented children who are buried in a degenerate school system and can't get out, the option is quickly emerging for parents to simply opt out. Not play and not have to deal with a broken system. I think we are on the verge of seeing many of our schools and even whole systems degenerate into being holding tanks/warehouses for truly dysfunctional youth with the rest opting for some form of neighborhood-based Internet-enabled home schooling.

Carole: I've seen ads that entice people to "find out if your spouse is having an online affair! Find out if your kids are surfing porn sites!" Any thoughts on the type of spyware that is used in the home?

Blaine: There is really not much difference between "home spying" and "corporate spying." It amounts to the bad guy wanting to violate a policy, and a system that is not adequate to support the policy. Probably one of the more significant overlooked notions is the word "personal" in the phrase "personal computer." The expectation of any protection in the out-of-the-box PC way outstrips the ability of the technology, particularly from an insider who has intimate access to the machine. Mostly this points to a serious lack of understanding of the technology. It really reduces to a fairly simple dictum: If you care about the information and the consequences of its misuse then, to the extent possible, eliminate the shared resource.

Carole: You stated that there are no "silver bullets." What is your opinion of vendors who are offering "one-stop shopping" for security services?

Blaine: The notion of "no silver bullet" is the notion that thus far there does not appear to be a single technology or single point of application for a technology that completely resolves the security challenge of most information systems. By that, I intend to point out that an IDS by itself is not, typically, a complete solution; PKI by itself is not a complete solution. The point is that security is a *system* problem and typically is not resolved through the introduction of a particular security service. Some vendors market a single product. Be cautious of vendors who argue that the single product is a complete solution. On the other hand, there are vendors who market suites of products that tend toward providing system-level solutions. These vendors are trying to provide one-stop shopping to their clients and, arguably, this could be a constructive approach. Arguably to the extent that the one-stop shops are dealing with the interactions and dependencies of the assorted products and understand the completeness of the solutions they offer. It's not a lot different from the notion of buying a car by the piece or as an integrated system. By the piece, one might get on the whole very high-quality individual parts, but one is now committed to dealing with the problem of assembling the parts into a whole. A great deal of energy will go into that effort and will require an organizational commitment to the continual maintenance of the whole parts-assembly business model. And it is not clear that all the parts go together to make something. However, by the car, one gets an integrated system that provides transportation, which is the overall objective.

Security is a *system* problem and typically is not resolved through the introduction of a particular security service.

Carole: What are your plans for the future?

Blaine: I would like to say something about this. The University of Nebraska at Omaha has offered me the opportunity to establish, build, and lead a Center for Information Assurance. We are committed to the mission of developing very skilled information-assurance professionals at both the undergraduate and graduate level. The center will be part of UN Omaha's College of Information Science and Technology and be housed in the University of Nebraska's Peter Kiewit Institute. We will develop a comprehensive undergraduate Information Assurance program targeted at supporting the Critical Infrastructure Protection Cybercorp initiative and developing the MS-level Information Assurance area of specialization. We are in the process of instrumenting a Security Technology Emulation and Assessment Lab. We are committed to developing the highly skilled and educated people, new knowledge, and appropriate technology to achieve a safe, secure, and reliable Information Age.

the bookworm

by Peter H. Salus

Peter H. Salus is a member of the ACM, the Early English Text Society, and the Trollope Society, and is a life member of the American Oriental Society. He is Editorial Director at Matrix.Net. He owns neither a dog nor a cat.

<peter@matrix.net>

BOOK REVIEWED IN THIS COLUMN

SECRETS AND LIES: DIGITAL SECURITY IN A NETWORKED WORLD

BRUCE SCHNEIER

New York: John Wiley, 2000. Pp. 412.
ISBN 0-471-25311-1.

This is an "extra" issue, so I want to break with tradition and discuss one (!) book.

Bruce Schneier's *Applied Cryptography* (1994; 2nd ed., 1996) is a truly splendid book. His new *Secrets and Lies: Digital Security in a Networked World* is really outstanding.

Schneier's byword is "Security is a process, not a product." Just as locking your apartment or your house (or your car) is a first step – not a solution – to the problems introduced by those few who want to prey on others' possessions, passwords, etc., are but a first step.

Schneier admits that he saw mathematics as a solution in 1994, but that he was wrong: cryptography (applied mathematics) doesn't exist in a vacuum; like everything else, we function within a highly complex environment. *Secrets and Lies* is an attempt at both describing the complexities of the digital environment and elucidating the methods available to render it more secure.

There are three parts to *Secrets and Lies*: The Landscape (with chapters on "Digital Threats," "Attacks," "Adversaries," and "Security Needs," pp. 11–81); Technologies ("Cryptography," "Cryptography in Context," "Computer Security," "Identification and Authentication," "Networked-Computer Security," "Network Security," "Network Defenses," "Software Reliability," "Secure Hardware," "Certificates and Credentials," "Security Tricks," and "The Human Factor, pp. 83–269); and Strategies ("Vulnerabilities and the Vulnerability Landscape," "Threat Modeling and Risk Assessment," "Security Policies and Countermeasures," "Attack Trees," "Product Testing and Verification," "The Future of Products," "Security Processes," and "Conclusion," pp. 271–395).

I happen to think security is important. It was while I was executive director of USENIX that we held the first security workshop (August 1988 in Portland, OR, chaired by Matt Bishop). Over the years I've reviewed a large number of books on security – ranging from Denning, Diffie, and Landau, to Bellovin and Cheswick; to Rubin, Geer, and Ranum and (last month) the new edition of *Building Internet Firewalls*. *Secrets and Lies* is up there with the best of them.

In fact, I think that Schneier has put the entire range of digital threats into appropriate context. I think that this is the book that every business executive should read. And it's written in a manner that every executive can understand. There's no code in it. No cryptographic algorithms.

There are lots of good examples and true stories.

In our increasingly digital world, the dangers need to be comprehended. Just as children need to learn how to cross the street, businesses need to know just how dangerous the networked world can be.

USENIX and SAGE news

Board Meeting Summary

by Gale Berkowitz
Deputy Executive Director
and Ellie Young
Executive Director

The following is a summary of some of the actions taken by the USENIX Board of Directors between June and August 2000.

Good Works

The Board voted to allocate $50,000 between now and 2001 for two programs sponsored by the Computing Research Association's Committee on the Status of Women in Computing Research. The first project is the Distributed Mentor Project *<http://www.cra.org/Activities/craw/dmp/index.html>*, in which outstanding female undergraduates work with female faculty mentors for a summer of research at the mentors' institutions. The second project is called Collaborative Research Experiences For Women (CREW) *<http://www.cra.org/Activities/craw/crew/index.html>*, where students work in collaborative teams with faculty mentors at their home institutions during the academic year.

SAGE

It was agreed that USENIX and SAGE will work toward coming up with a model that gives greater autonomy to SAGE.

International Affiliate Membership Category

The USENIX Board of Directors voted to accept a proposal for a second international affiliate membership category. Affiliate members will have all the same membership benefits as an individual member except voting privileges, and will receive access to *;login:* in pdf format through the Affiliate Group's members-only Web site.

Bylaws and E-voting

A committee was formed to review the USENIX bylaws and amend them to allow us to conduct elections electronically.

Conferences

It was agreed that the Windows Systems Symposium will no longer be held and that the calls for papers for other USENIX events should encourage papers from all platforms and operating systems. A system administration of Windows conference might be held depending on support from SAGE and Microsoft.

A file-systems storage conference, chaired by Darrell Long, was approved.

It was agreed that USENIX will cosponsor the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV).

Press Releases

Board members were tasked with writing position papers/issuing press releases on the implications of new technologies e.g., electronic voting.

Next Meeting

The next meeting of the Board of Directors will be held December 5, 2000, in New Orleans, LA.

SAGE Elections

Elections for the SAGE Executive Committee for the 2001-2003 term are coming up soon. For the first time, voting will be conducted electronically. VoteHere.net has been selected to conduct the elections.

To be eligible to vote, you must be a SAGE member on December 1, 2000. Since voting will take place electronically, it is essential that your membership information is up to date, particularly your email address. To vote, you will need your membership number and password.

To verify and update your membership information, please go to: *<https://db.usenix.org/membership/update_member.html>*.

Election notifications and candidates' statements will be available on the SAGE Web site (*<http://www.usenix.org/sage/election01/>*) by December 11, 2000.

Notifications and voting instructions will be sent via email to all current SAGE members. For those who choose to not submit their ballots electronically, a paper ballot option will be made available through the voting website.

To find out more about the candidates running for seats on the Executive Committee, please attend the Candidates' Forum being held on December 7, 2000 at LISA in New Orleans. Candidates' statements will also be available through the SAGE website.

This is another great reason to register early for LISA, and be sure that your membership is up to date in time for the elections. The pre-registration discount deadline for LISA is October 27, 2000.

For more information about SAGE governance, please see: *<http://www.usenix.org/sage/official/>*.

USENIX Good Works Program

Every year income from the USENIX endowment fund and our conferences are used to help nurture the development of the advanced computing systems community. In 1999 USENIX spent over a million dollars on such good works. Here are some details.

USENIX Student Programs

Graduate and undergraduate college education is always of the highest priority to the Association. USENIX and its members value students and the research in the computing systems arena that is generated in colleges and universities. Recognizing the importance of this work, USENIX generously funds a number of programs for college students: stipends for students to attend USENIX and SAGE conferences, scholarships, student research projects, outreach to representatives on campuses, as well as several innovative, computing-related projects. The student stipend program offers travel grants to enable full-time students to attend USENIX conferences and symposia. Over 360 institutions have been represented in the USENIX Student Stipend Program. To date, over 100 schools have designated outreach representatives. Our Scholastic Program provides funding for scholarships and student research projects. More information about our student programs, at: <http://www.usenix.org/students/students.html>.

Computing Community Projects

The USENIX Association is pleased to announce the funding of two important projects that are relevant to the USENIX and SAGE communities: The Internet Software Consortium BIND v9 project, and the Electronic Frontier Foundation's legal work for two important cases, the Bernstein encryption software case, and DVD DeCSS cases.

USENIX, with Stichting NLnet Foundation, has also launched the Research eXchange, an international research exchange initiative for computer software-related networking technologies, called ReX. Information is at <http://www.usenix.org/about/rex.html>.

Here is a list of other projects USENIX funded this year:

Travel stipends for the African Network Infrastructure Meeting in Cape Town in May.

Student stipends for travel and registration to attend the Computers, Freedom and Privacy Conference and the Fast Software Encryption Workshop.

Incident Cost Analysis and Modeling Project (I-CAMPII) of the University of Michigan to study the frequency and costs of IT-related incidents.

Software Patent Institute (SPI) to expand, and improve SPI's Database of Software Technologies.

SOS Children's Village Illinois, to support the purchase of computers and network hardware and software for this non-profit foster care agency.

Sponsor the USA Computing Olympiad for high-school students.

Women in Computing

USENIX is dedicated to increasing the representation of women in the computing professions. In our efforts to support women's fuller participation, USENIX has contributed to funding the production of a video targeted at high school and college students. The video "Career Encounters: Women in Computing" has been broadcast nationally on cable and satellite public television networks. For information about this video, visit <http://www.davisgrayinc.com/new1.html>.

USENIX will also be providing support for two programs sponsored by the Computing Research Association's Committee on the Status of Women in Computing Research. The first project is the Distributed Mentor Project (<http://www.cra.org/Activities/craw/dmp/index.html>), in which outstanding female undergraduates work with female faculty mentors for a summer of research at the mentor's institution.
The second project is called the Collaborative Research Experiences For Women (CREW) (<http://www.cra.org/Activities/craw/crew/index.html>), whereby students work in collaborative teams with a faculty mentor at their home institution during the academic year.

USENIX is proud to be a sponsor of the recent Grace Hopper Women in Computing conference.

For more information about the USENIX Good Works program, please see: <http://www.usenix.org/about/goodworks.html>, or contact Gale Berkowitz, Deputy Executive Director, at <gale@usenix.org>.