Hal Berghel

Faith-Based Security

A tongue-in-cheek look at serious security issues.

IT security has received increased attention primarily, but not exclusively, due to the increased threat from viruses, worms, password crackers, Trojan horses, and a cornucopia of other types of malware and exploits. As a consequence of this increased attention, a variety of security models have been proposed. Security in depth (SID) is one such example. Winn Schwartau's time-based security is another. In this column, I offer another modest example extrapolated from popular culture: faith-based security, aka "no network left behind."

HAL MAYFORTH

Security Models

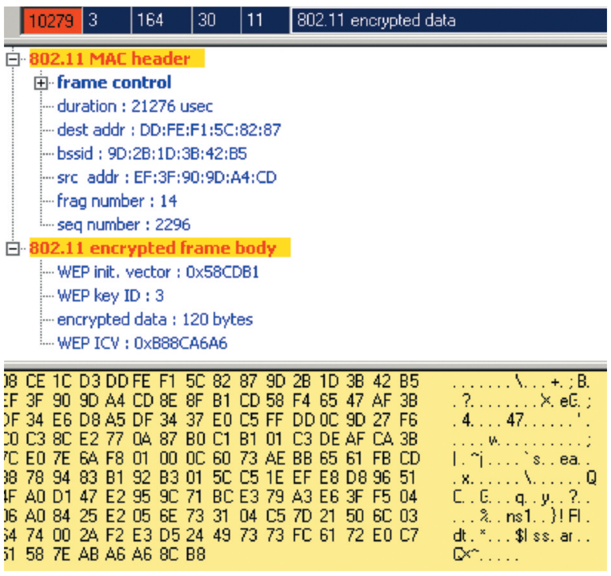
By their very nature, security models are usually out of date. Security modeling is akin to driving forward while looking through the rearview mirror since security systems are primarily reactive. The problem is illustrated by zero-day exploits where the first appearance of an exploit coincides with the first appearance of a vulnerability. One of the grand challenges in future digital security is to figure out how to model the unknown in anticipation of post-modern exploits, such as zero-day attacks and so-called "super worms."

Security models also tend to be obtuse. Though "security in depth" is a common phrase in IT circles, few could define it precisely. The phrase has been used to describe everything from cascaded network defenses and layered intrusion prevention/detection systems to differentiated password-control policies. About the only common theme I can detect is that security-in-depth seems to be used interchangeably with "more is better."

The Security in Depth Fallacy

There is an interesting fallacy in informal logic called the principle of vacuous alternatives. It goes something like this: Take any sentence. If the negation of that sentence seems preposterous, then the original sentence is likely vacuous. As an example, consider "I believe in justice." The negation, "I don't believe in justice," seems like an absurd remark. It's not that it's nonsensical. Rather, it has no conversational contribution to make as it's difficult to imagine how any reasonable person could disagree with it. Vacuous propositions behave like semantic tautologies.

Such is the case with security in depth. Have you ever heard an IT professional champion the cause of "superficial security," "shallow security," or "myopic security?" Not likely. This is the primary reason why security in depth is so poorly understood. Its vagueness quickly gives way to vacuousness on inspection.



```
10279 | 3 | 164 | 30 | 11 | 802.11 encrypted data

⊟ 802.11 MAC header
   ⊞ frame control
   duration : 21276 usec
   dest addr : DD:FE:F1:5C:82:87
   bssid : 9D:2B:1D:3B:42:B5
   src  addr : EF:3F:90:9D:A4:CD
   frag number : 14
   seq number : 2296
⊟ 802.11 encrypted frame body
   WEP init. vector : 0x58CDB1
   WEP key ID : 3
   encrypted data : 120 bytes
   WEP ICV : 0xB88CA6A6

8 CE 1C D3 DD FE F1 5C 82 87 9D 2B 1D 3B 42 B5    ........\...+.;B.
F 3F 90 9D A4 CD 8E 8F B1 CD 58 F4 65 47 AF 3B    .?........X.eG.;
F 34 E6 D8 A5 DF 34 37 E0 C5 FF DD 0C 9D 27 F6    .4....47.......'.
0 C3 8C E2 77 0A 87 B0 C1 B1 01 C3 DE AF CA 3B    ....w...........;
C E0 7E 6A F8 01 00 0C 60 73 AE BB 65 61 FB CD    |.~j....`s..ea..
8 78 94 83 B1 92 B3 01 5C C5 1E EF E8 D8 96 51    .x......\......Q
F A0 D1 47 E2 95 9C 71 BC E3 79 A3 E6 3F F5 04    C..G...q..y..?..
6 A0 84 25 E2 05 6E 73 31 04 C5 7D 21 50 6C 03    ...%..ns1..}!Pl.
4 74 00 2A F2 E3 D5 24 49 73 73 FC 61 72 E0 C7    dt.*...$Iss.ar..
1 58 7E AB A6 A6 8C B8                            QX~.....
```

Figure 1. The WEP initialization vector is communicated in cleartext (for example, 0x58CDB1).

Security Through Obscurity

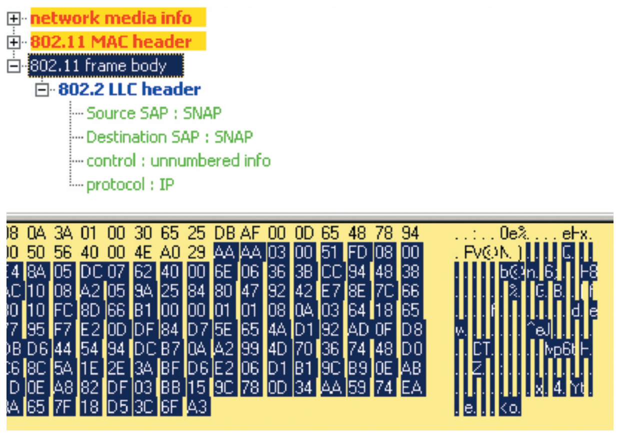
I admit that a prima facie case could be made for security in depth even in the naive sense of "more is better." When I propose adding a new vitamin to my diet, my internist tells me "At this point there is no physiological evidence that suggests that this substance is harmful to humans, so knock yourself out." As with my vitamins, a random application of security applications and systems is unlikely to do any more harm than lure one into a false sense of security and perhaps slow things down a bit. And like the vitamins, when carefully and judiciously applied and evaluated in a controlled experimental setting, even naive security in depth can be of some value.

Such is not the case with our third model: security through obscurity (STO). No prima facie case may be made here. The general premise of STO is that inviolability is a consequence of the enigmatic. This is same sort of reasoning that helped the Imperial Japanese Navy and German Wehrmacht become the global powers they are today. The Japanese Purple and JN-25 codes and the German Enigma cipher system were assumed to be inviolate precisely because of their hidden complexity. As far back as the 1880s, Auguste Kerckhoffs proposed that no cryptographic system that purports to be secure should be predicated on the assumption that no one would ever figure out how it worked—rather the emphasis should be robustness of the procedure and key strength. Both Axis powers failed to comprehend the weakness of STO. This also speaks in favor of the robustness of open source software.

Despite our intuitions, many software systems have adopted STO to their cost. To illustrate: Windows buffer overflows, such as the IDQ.DLL overflow in the Code Red Worm, were entirely predictable to anyone who knew how the Windows ISAPI extensions worked. This was a design defect that produced a buffer overflow and ran the malware with elevated privileges since IDQ.DLL runs within Inetinfo.exe as local administrator. It was assumed no one would notice the inadequate bounds and error checking built into the operating system. We'll place this in STO category I: failure to write secure code. Conceptually similar vulnerabilities, like format string attacks (printf) in Unix and SQL compromises in Windows (IIS/RDS), would also fall into our first category.

Another example is the entire suite of 802.11 security vulnerabilities. In this case, the defect was actually built into the standards. Nowhere is this more evident than with the wired equivalent privacy (WEP) protocol.

WEP has many "issues" that go beyond our current interest. However, one stands out as a paradigm case of a mistake carried through to perfection: the sloppy implementation of the RC4 symmetric, stream cipher. The faulty WEP algorithm was a part of the original IEEE 802.11 protocol



```
⊞ network media info
⊞ 802.11 MAC header
⊟ 802.11 frame body
   ⊟ 802.2 LLC header
      Source SAP : SNAP
      Destination SAP : SNAP
      control : unnumbered info
      protocol : IP
```

```
08 0A 3A 01 00 30 65 25 DB AF 00 0D 65 48 78 94    ..:...0e%....eHx.
00 50 56 40 00 4E A0 29 AA AA 03 00 51 FD 08 00    .PV@.N.)
E4 8A 05 DC 07 62 40 00 6E 06 36 3B CC 94 48 38
AC 10 08 A2 05 9A 25 84 80 47 92 42 E7 8E 7C 66
80 10 FC 8D 66 B1 00 00 01 01 08 0A 03 64 18 65
77 95 F7 E2 0D DF 84 D7 5E 65 4A D1 92 AD 0F D8
0B D6 44 54 94 DC B7 0A A2 99 4D 70 36 74 48 D0
C6 8C 5A 1E 2E 3A BF D6 E2 06 D1 B1 9C B9 0E AB
0D 0E A8 82 DF 03 BB 15 9C 78 0D 34 AA 59 74 EA
AA 65 7F 18 D5 3C 6F A3
```

Figure 2. The 802.11 frame body always begins with a SNAP header (for example, AA).

specification.

Generally, WEP works like this. The RC4 algorithm uses the pseudorandom generation algorithm (PRGA) to produce a key-stream of bits that are XORed with the plaintext to create the ciphertext. Key-change is accomplished by adding an Initialization Vector (IV) that makes each packet key unique. The IV is concatenated with the WEP key to form the WEP seed.

The properties of the IV are interesting:

1. The IV is only 24 bits long;
2. The IV is always prepended to the WEP key;
3. The IV is always transmitted in cleartext (see Figure 1);
4. Some IVs are "weak" in the sense that they suggest information about the key—the first bytes of a typical WEP packet are typically the snap header 0xAA (see Figure 2);
5. The IEEE standards were so ambiguous that many vendors used sequential IV generators that begin with 00:00:00 and wrap with FF:FF:FF; and
6. The key-generation algorithm itself is hobbled because the most significant bit of each key is always 0; thus it only produces unique keys for seeds 00:00:00:00 through 00:7F:7F:7F.

The community of FMS (after Fluher, Mantin, and Shamir) attack analysts reacted immediately. In short order a flurry of successful WEP-cracking tools were developed (WEPAttack, WepCrack, Aircrack, WepLab, WEPWedgie) all made possible by the faulty implementation of RC4. A virtual cottage industry was made possible because the original WEP security standard followed the STO model. We will put the WEP vulnerability into our new STO Category II: botched implementations.

One might think the frailty of WEP would have triggered a total rethinking of WiFi security. Such is not the case. While WEP's successor, Wireless Protected Access (WPA), did strengthen the integrity-checking algorithm and key management, it basically just added another layer of obscurity over the sloppily designed WEP in the form of a shell over the RC4 algorithm. Deployed by the Wi-Fi Alliance in 2002, WPA didn't really eliminate the key-management problem inherent in WEP, but rather proliferated the number of keys involved. WPA uses a pairwise master key (PMK) to generate additional keys that are combined with sender MAC address, packet sequence number, the wireless Service Set ID, and SSID length as grist for the hashing mill (PKCS #5 v. 2.0). Let's think about this. If an underlying procedure is faulty, does it become less faulty if we use it over and over and over again? WPA relied on STO, just like its predecessor. Predictably, within a year of release, a successful WPA attack was discovered. Shortly therafter, the WPA-cracking utility coWPAtty was released that reverse engineers the PMK from the SSID, SSID length, and sequence number MAC address, and WiFi security was back at the starting block.

Neither was the Extensible Authentication Protocol immune. Cisco's version of EAP, LEAP, deserved the term lightweight. LEAP's major fault was that it relied on the MS-CHAPv2 hashing algorithm for authentication. MS-CHAPv2 does not use "salt," so the same plaintext value will always produce the same hashed value. This makes EAP-LEAP vulnerable to dictionary and replay attacks. Once again, the defense of EAP-LEAP ultimately relied on no one finding out how the system works. Auguste Kerckhoffs could have predicted this without ever seeing a computer.

My final example came to my attention in the past few weeks. MIFARE is a proprietary encryption technique for RFID (radio frequency identification) developed by Philips and Siemens in the late 1990s. MIFARE is an attempt to cryptographically secure the now-ubiquitous RFID space that relies on RF transmission for communication between transmitter and receiver.

Following the common theme, the security of the proprietary MIFARE system is predicated on the belief that no one will discover how it works. And, as one might predict, some MIFARE circuits were reverse engineered down to the gate level. The result was the discovery that the random number generation that drove the encryption resulted from a 16-bit key linear feedback shift register based on a master key and a time signature. With RFID sniffing via an open PICC (proximity integrated contactless chip) card and a logic analyzer, it is possible to discern patterns in the challenge-response authentication procedure that can be used in a replay attack, and from there it is possible to recover the key from the value of the unique identifier and the observed behavior of the shift register in the authentication process. We'll create STO category III for this MIFARE vulnerability: turning chip designers loose with CAD/CAM software without adequate education and training.

URL PEARLS

When it comes to digital security systems, secrecy is indeed the mother of dysfunction. The security vulnerabilities described in this column were real and betray only the slightest hint of literary hyperbole. For those interested in the details, two of the security-through-obscurity examples were covered in previous columns: the Code Red Worm was discussed in December 2001; and Wireless Infidelity appeared in December 2004 and again in August 2005. The RFID MIFARE exploit was presented at the 24th Chaos Communication Congress last December (see events.ccc.de/congress/2007); a video of the presentation by Karsten Nohl and Henryk Plotz is available at video.google.com/videoplay?docid= 4252367680974396650&hl=en. C

Faith-Based Security

Examples of failed STO could fill a weighty tome. I've mentioned three. These examples highlight the consequences of building deficiencies into the design of things or at least unwittingly including them. The flaws would likely have been detected and reported had the code, system, or chipset been carefully analyzed during impartial peer review by qualified professionals.

But I don't want to leave this critical view of deficiencies at the feet of naive SID or STO. I'm looking for first principles here.

I'll refer to the common elements between them under the general rubric of faith-based security—in the most secular sense of this popular phrase. The only thing these two security models have going for them is the unsupportable and unjustified faith that they are reliable. These are manifestations of the technologist succumbing to the self-deception that secrecy and tight lips will cover all design misjudgments.

I propose that faith-based security enter our vocabulary as the default model of IT security. Let's get the faith-based orientation of naive security in depth and STO up front where it belongs. Think of the advantages. If an auditor asks why we decided to place our Web server on the inside of our enterprise firewall, we report that we have faith in our Internet comrades. Faith is a predicate of propositional attitude, like belief, want, and desire. If someone says they have faith in something, one can't say "No you don't," at least not until someone comes up with a method to read thoughts. The auditor doesn't have faith, we do have faith; half-empty, half-full. You get the idea.

Since the integrity of a faith-based security implementation is by definition taken on faith, we hold the position that whatever policies and procedures discovered by an auditor were actually intended. So what if our corporate mailer is running on an operating system that hasn't been supported since perestroika—we have faith in good old "digital iron." After all, when was the last time you read about some hacker compromising OS/2 or Multix? So the primary remote access to our file server is TFTP; our spin is that any protocol that old is "time-tested." So our password security policy requires LAST-NAME followed by YEAR; we emphasize that we have a rule for password expiration built right into our password security policy.

No baselines to measure, no checklists to distract us, no concern over best practices, no specific objectives to define. COBIT? Out the window. FISCAM? Who needs it? SOX, HIPAA, GLB? No thank you.

So the next time someone challenges your organization's security model, rather than beating around the bush, making excuses, blaming budgetary woes, faulting management's lack of vision, or chastising vendors, think outside the box. State up front that your security model is faith-based and take a swerve around all the minutiae. Treat these details like all of those log files you haven't reviewed since you upgraded to NT Service Pack 2. Build in backward "time basing" to the ultimate IT apocalypse—the implosion of the commercial Internet. After that, who will care about digital security anyway? C

Hal Berghel is associate dean of the Howard R. Hughes College of Engineering at the University of Nevada-Las Vegas, the director of the Center for Cybersecurity Research (ccr.i2.nscee.edu), and co-director of the Identity Theft and Financial Fraud Research and Operations Center (www.itffroc.org).

DOI: 10.1145/1330311.1330315

Coming Next Month in COMMUNICATIONS OF THE ACM