A Case for Exploit-Robust and Attack-Aware Protocol RFCs

Venkat Pothamsetty[1] and Prabhaker Mateti[2]

[1]Cisco Systems
Critical Infrastructure Assurance Group
Austin, TX 78727, USA
vpothams@cisco.com

[2]Wright State University
Dept. of Computer Science and Engineering
Dayton, OH 45435, USA
pmateti@wright.edu

Abstract

A large number of vulnerabilities occur because protocol implementations failed to anticipate illegal packets. RFC*s typically define what constitute "right" packets relevant to the protocol and they specify what the response should be for such packets. They are often ambiguous and remain silent on what the protocol implementation should do for packets which deviate from the specification.*

Implementers must and, by and large, do faithfully implement an RFC*. However, implementers usually take any silence in a specification as "design freedom". Even though the protocol implementers are network specialists, they often are not knowledgeable in network security and cryptography issues, past exploits and common attack techniques that can impact the security of a protocol module, and consequently, the whole system.*

This paper systematically discusses vulnerabilities that can be attributed to protocol designs, inadequacies of RFC*s, and omissions of the protocol implementers. Using specific examples, we point out how ambiguities in protocol* RFC*s have lead to security vulnerabilities. We correlate various types of security vulnerabilities with the way the* RFC*s are written. We make a case for such exploit-robust and attack-aware* RFC*s, and recommend the features for a better* RFC*, called e*RFC *(Enhanced* RFC*). We offer advice to* RFC *writers, implementers and* RFC *approval bodies. The most effective solution to reducing network security incidents is to fix the* RFC*s in such a way that the implementers are forced to write an exploit-robust implementation, irrespective of their security knowledge and expertise.*

1 Introduction

Attack techniques are becoming more sophisticated in exploiting ambiguous protocols and their fragile implementations (see [3], and [8]). Protocol RFCs continue to be written in an ambiguous manner, ignorant of the security and attack technique trends, and corresponding implementations continue to be vulnerable. By doing so, the RFCs routinely commit the "Seven Sins of the Specifier" [5].

1.1 Security Robustness

Security robustness is generally well-understood, but has not been well articulated. Here is what it means to us. (These issues are considered in greater detail and precision in [4].) (i) No matter what the packets are, a protocol engine must not crash or hang. (ii) Serve only legitimate packets. (iii) Do not send an invalid response to a valid request from a legitimate client and user combination. (iv) Do not send a valid response to a bogus request. (v) Do not delay, beyond computing power and network traffic limitations, sending a response unless it is never to be sent.

1.2 Specifiers'/Implementers' Problem?

Industry has few options to counter the problem of non-robust implementations. Static analysis tools [6] cannot effectively catch vulnerabilities arising due to inherent protocol design problems, nor do they point out vulnerabilities arising due to common attack techniques. Security education of network software engineers is an effective solution, but the solution is as good as the implementer is.

The most effective solution is to fix the RFCs. The specifications should be written in such a way that if the implementer follows the specification (he will by contract), the resulting implementation will automatically be robust, withstand common attack techniques and will have incorporated security measures.

Furthermore, it is much easier to educate relatively fewer number of specifiers than an uncountable and scattered implementers. We acknowledge that some of the attacks are dependent on platform specific resource limitations, which the specifier has no control over. However, in such cases, we recommend that the specifier explicitly acknowledge the issue, and recommend implementation considerations to the implementer.

1.3 Security Considerations in RFCs

An RFC[1] must (now) have a security considerations section[2] for it to be accepted. The security considerations section is a good start; however, it suffers from the following.

A single security considerations section does not effectively address attacks that could happen at various stages during the protocol. For example, attacks (such as message insertion, and flooding), security techniques (such as authentication) and security measures (such as error recovery) depend on the state that the protocol is in. We believe that the threats could be effectively addressed in relevant sections all through the RFC, and not just at the end.

Most RFCs do not (thoroughly) follow RFC3552 guidelines. Consider the following selected examples. (i) RFC3928[3] does not methodically consider all attack scenarios. It merely considers flooding attack and ignores the rest. (ii) RFC3937[4] claims, without justifying, that the RFC brings no additional threats. (iii) RFC3952[5] identifies a potential denial of service (DoS) but does not analyze how the protocol counters that threat.

1.4 Ambiguity of Technical Prose

RFCs are written up as technical reports in English. This is prone to ambiguities and misinterpretation. It is a well-known but perhaps unacknowledged fact that different network protocol stacks would not be interoperable had it not been for the wide accessibility of source code that implemented the early and "infrastructure" protocols such as TCP. Even today, one can safely predict that, if an implementation team is given only the RFCs and denied all access to source code of any implementations, the result would not be interoperable.

1.5 Historic Design and Intent of RFCs

A part of the ambiguity arises because of the words the RFCs chose as building blocks of the RFC - "Must", "Should" and "May" (see RFC2119). Historically, the design intent of the RFCs has been to interpret the packets leniently, and words like "may" are deliberately intended to allow protocol implementations to be lenient in the interpretation.

However, for a protocol implementation to be secure, it must be strict in interpretation. Many a vulnerability (e.g., buffer overflows) today are due to sloppy implementations which do not *validate* the packets before parsing them. We argue that the problem however is that the implementers do not have a robust specification against which they can *validate* their implementation.

2 Inadequate RFCs

This section discusses various "sins" of RFCs such as being silent and/or ambiguous and how they contributed to security vulnerabilities.

2.1 Header and Data Values

Quite a few RFCs are ambiguous or silent about the range, type and size of valid header values. Though some of these valid values are derivable, the not-so-direct specification of header values result in an implementation not checking for the type and values inside the packet headers before parsing.

2.1.1 Valid Values of Fields

A classic example is how RFCs specify the *length* field of a header. E.g., ISAKMP RFC2408 has this on page 24: "Length (4 octets) - Length of total message (header + payloads) in octets". However it leaves many questions (whose answers need to be deduced) such as: Can the payload be of zero bytes? What is the minimum size of payload? What is the maximum possible length of the payload? In the absence of such specification, the

[1] All RFCs are available at `www.rfc-editor.org`.

[2] RFC3552: Guidelines for Writing RFC Text on Security Considerations (2003)

[3] Lightweight Directory Access Protocol (LDAP)

[4] A Uniform Resource Name (URN) Namespace for the International Press Telecommunications Council (IPTC), Oct 2004, Informational

[5] Real-time Transport Protocol (RTP) Payload Format for internet Low Bit Rate Codec (iLBC) Speech, Dec 2004, Experimental

implementer will try to take the value as-it-is and try to interpret the rest of the packet as per the supplied value, leading to a multitude of vulnerabilities. E.g., implementation of ISAKMP daemon in OpenBSD had vulnerabilities because the length of the received packet is not validated against valid and possible data values[6].

Had the specification been written (e.g., by specifying the min and max values of length) so that the implementer is forced to ascertain the value before parsing, we believe that the vulnerability could have been avoided. Scores of vulnerabilities[7] of this type make us advocate that the specification should explicitly and directly mention the range of valid header values that the implementation should check before parsing the values.

2.1.2 Valid Types

RFCs are typically ambiguous and/or silent about the valid *type* of header or data values of a protocol packet. In cases where the value of a header or data must have only specific types of values (strings in the case of email address, e.g.), we argue that RFCs should explicitly specify the type and bind the implementer to check the validity of the type of values before parsing.

MIME protocol: Many vulnerabilities exist in MIME protocol (RFCs 2045-2049) because different vendors interpreted RFCs comments on interpreting characters in different ways.[8] The corresponding vulnerabilities become more dangerous when the characters have special meanings, such as the back-slash and null characters in web applications.[9] [10]

2.1.3 Valid Sizes

RFCs do not typically specify the valid size of the accompanying payload. Many vulnerabilities arise because the implementations try to parse the entire payload supplied.

Telnet Options: Within every BSD derived `telnet` daemon under UNIX, the telnet options are processed by the `telrcv` function. This function parses the options according to the telnet protocol and its internal state. During this parsing the results which should be sent back to the client are stored within the `netobuf` buffer. This is done without any bounds checking, since it is assumed that the reply data is smaller than the buffer size (which is `BUFSIZ` bytes, usually).[11] The corresponding buffer overflow has lead to multiple vulnerabilities.[12] Had the Telnet RFC854 specified explicit limits on the size of the options[13], we believe that this vulnerability could have been avoided.

SSL: There are a few vulnerabilities in SSL because SSL did not put a limit on the data portion (client certificates[14], e.g.)

2.1.4 Actions for Illegal Packets

RFCs do not exhaustively specify the actions (or series of actions) that the implementation needs to take in case any of the values of the headers inside the packets it resides are not compliant with the protocol. The actions may be (a) drop the packet, (b) close the connection, (c) send an error packet or (d) any combinations of these actions. In the absence of such conditions, the implementer will be free to interpret such packets. In many cases the implementer will continue to interpret, store the information, and pass on the packet to other modules causing vulnerabilities.

TCP: RFC793 and others in defining the functional specification for TCP do define how systems should respond to legitimate packets, but they don't explain how systems should handle illegal combinations of flags. Consequently, different operating systems respond differently to illegal flag combinations. Attackers now exploit this to finger print the operating system.[15]

SNMP: Many of the vulnerabilities indicated by the test cases released by `www.ee.oulu.fi/research/ouspg/` for SNMP protocol are because the corresponding implementation did not have code to handle header values that are inconsistent with the protocol specification.[16]

2.2 Pre And Post Conditions

The RFCs are typically ambiguous/silent/non-exhaustive about the conditions that need to be fulfilled before entering a protocol state and after leaving a protocol state. Many vulnerabilities exist because of not-so-strict pre and post conditions in RFCs. The following are but two examples:

Unauthorized Access: If the implementation does

[6]xforce.iss.net/xforce/xfdb/15518

[7]CAN-2005-0340, CAN-2005-0482, CAN-2004-1002, CAN-2004-0899. These and others can be viewed at www.cve.mitre.org/cve/.

[8]www.niscc.gov.uk/niscc/docs/al-20040913-00537.html

[9]www.webappsec.org/projects/threat/classes/path_traversal.shtml

[10]nessus.org/plugins/index.php?view=single\&id=12124

[11]www.monkey.org/openbsd/archive/misc/0107/msg01332.html

[12]www.cert.org/advisories/CA-2001-21.html

[13]http://www.iana.org/assignments/telnet-options

[14]archives.neohapsis.com/archives/bugtraq/2002-02/0313.html

[15]www.securityfocus.com/infocus/1200/\%3E

[16]www.cert.org/advisories/CA-2002-03.html

not reset all the variables of a session, it might be possible for the next user to gain unauthorized access to the prior variables (related to the previous user). [17]

Resource Lockup: If the implementation does not release the resources (such as memory) used by a previous state, it might lead to resource exhaustion issues, such as NNTP service in Windows contains memory leak.[18]

We summarize the recommendations of [7]. (a) For secure state transitions, we must specify all the pre-conditions for every state. (i) Trust pre-conditions that needs to be met, such as authentication and authorization conditions that need to be met before entering a state. (ii) Resource pre-conditions that needs to be met, such as amount of memory that must be available before entering a state. (b) The specification needs to specify all the post-conditions of each state. (iii) At state hand-over, all resources allocated to the connection should be reclaimed. (iv) At state hand-over, all changes that are made to the system should be reset/undone. (c) Miscellaneous conditions such as the following also must be stated. (v) The connections at later stages should not be affected by the connections at preliminary stages. (vi) The connections should have timeouts and a secure exit strategy, including either handing over the connection to the previous state or terminating the connection. (vii) A connection at an earlier state should not be able to preempt a connection at a later state without the proper credentials.

2.3 Timeouts

Timeouts, the time that the implementation has to wait (usually for an event such as packet arrival) in a particular protocol state, play a very significant role in influencing the security of a protocol. RFCs are frequently ambiguous and/or silent about the timeouts. RFCs either ignore timeouts or specify too long a timeout. IKE RFC2409, for example, does not specify timeouts for connections in various modes, including "Main Mode" and "Quick Mode".

The following examples of vulnerabilities make a case for RFCs to explicitly state reasonable timeouts for each state.

Denial of Service: If a client does not reset a session (e.g., the client connects to a session and be idle), it might effectively cause Denial of Service on other legitimate users. Windows Telnet daemon had such vulnerability.[19]

[17] www.securityfocus.com/bid/9807/discuss

[18] www.securiteam.com/windowsntfocus/5EP0B1F550.html

[19] www.securityfocus.com/bid/2018

Resource Exhaustion: Since a protocol state is essentially a memory dump of state variables, too large timeouts can cause memory consumption on routing/ fire-walling devices. Several fire-walling devices were vulnerable because of the above described scenario.[20]

2.4 Number of Headers and Header Fields

Many protocols have a provision for sending varying number of headers (typically to advertise the protocol's capabilities, so that the other end of the protocol can choose the appropriate one).

RFCs typically do not specify limitations on the number of such inner headers an implementation should entertain. Because of this silence, the implementation often continues to parse headers leading to vulnerabilities. Some examples are listed below:

RADIUS: Each RADIUS (RFC2865) packet can be up to 4096 bytes. It allows packing more than 2000 RADIUS attributes into a single packet. Some RADIUS server implementations allocate maximum possible attribute length for each attribute, that is, for each attribute up to 256 bytes of memory will be allocated. It is thus possible to lock about 512K of memory and waste CPU time with a single 4K packet resulting in an easy denial of service attack.

ISAKMP: An ISAKMP (RFC2408) packet with a malformed delete payload having a large number of SPIs will cause ISAKMPD to read out of bounds and crash.[21] We believe the vulnerability would have been avoided if the ISAKMP RFC mandated limits over the acceptable number of SPIs.

3 Knowledge of Security Issues

It is not mandatory that RFC writers propose an exhaustive set of security mechanisms for a newly proposed protocol. The following sections make the case that various security mechanisms must be thoroughly investigated and proposed along with the protocol specification. Note that some of the following sections reinforce the security considerations guideline RFC3552 through examples of past vulnerabilities.

3.1 Confidentiality and Integrity

The non-cryptographic RFCs (as opposed to cryptography-based RFCs like IPSec) typically do not specify any mechanisms for protecting protocol communication.

[20] www.kb.cert.org/vuls/id/539363

[21] CAN-2004-0221

The H323 protocol suite and SIP (RFC3261 Session Initiation Protocol) protocols did not think deeply about cryptographic issues (encrypting the signal and media traffic) when the RFCs are written. As a result, it took quite a while for the industry to figure out what the right encryption mechanisms are for Voice-Over-IP traffic.

3.2 Authentication

The network infrastructure community has seen many vulnerabilities in the initial versions of routing protocols because of lack of authentication and had to include authentication mechanisms in later versions of the protocols. The following are some examples:

RIP v1: RIP v1 (RFC1058) was vulnerable to spoofing.[22] RIP version 2 (RFC1723 and RFC2082) includes plain text and MD5 authentication mechanisms.

OSPF and BGP: These had to do go through the same phases, they needed to be extended to include authentication mechanisms. Later OSPF (RFC2328) and BGP (RFC2385) RFCs have authentication mechanisms, but it is unclear how many installations actually use these authentication mechanisms for the protocol does not mandate the use of those authentication mechanisms.

Original SIP: RFC3261 employed HTTP authentication mechanism for request messages but not for registration messages. Therefore, registration message is spoofable. The RFC is being currently revised to include digest mechanisms.

Many of the newer protocol specification writers are considering authentication mechanisms. iSCSI [2] is an example of such RFCs. However, given the above history it is prudent to mandate RFCs to address the security mechanisms in the first draft itself.

3.3 Authorization

RFCs do not typically analyze the authorization issues that will crop up when a protocol has been implemented and the corresponding application is deployed in large scale.

E.g., H323 protocol suite and SIP recommend opening up random ports for transferring media, and it has become hard for firewalls to employ fix-ups (so that only authorized persons have access to that traffic) and secure the corresponding traffic.

[22]CVE-1999-0111

4 Attack Techniques

Designs of (future) protocols must take into account attack techniques. Judging from the RFCs, protocol specifiers are unaware of attack techniques and therefore cannot specify mechanisms against them.

4.1 Flooding

Flooding related vulnerabilities arise because the implementations failed to anticipate an abnormally high number of valid packets within a short duration. RFCs typically do not specify the actions that the implementation need to be taken when it receives a flood of packets as part of a protocol communication.

We have seen the many implications of this silence. The vulnerabilities related to SYN floods and ARP floods are because the specification failed to anticipate such packets. Irrespective of these classic attack patterns, we have seen RFCs continuing to be silent on this aspect. E.g., IKE RFC2409 failed to anticipate a flood of initial (Main Mode) packets, and the implementations continue to be vulnerable[1].

4.2 Out of Sequence

The TCP RFC793 did not specify re-assembly timeouts of out-of-sequence numbers, and IP RFC791 did not specify fragmented datagrams, which lead the implementers follow different waiting schema for packet re-assembly. Multiple memory exhaustion vulnerabilities[23] can be attributed to such implementations.

Such a pattern of specification continues to exist. E.g., ISAKMP RFC2408 did not specify the sequence of payloads and BSD ISAKMPD suffered.[24]

4.3 Sniffing

It is no longer reasonable to think that packet contents are not sniffed. Encrypted packets add computational burden of decyphering not only to the receiving hosts but also to the sniffer (who may not possess the relevant keys and hence must resort to brute-force). Note that there is enormous pressure on making receiving hosts (especially on embedded devices such as IP phones and wireless access points) inexpensive, implying that their computational powers are meager. Depending on the nature of the protocol, the trade-off between leaving the packet unencrypted (fast processing of unencrypted packet contents together with loss

[23]`attrition.org/security/advisory/idefense/idefense-04-03-02.freebsd`

[24]`openbsd.groupbsd.org/errata31.html\#isakmpd`

of confidentiality) versus encrypted packets must be evaluated.

4.4 Spoofing

Nearly all of the protocols designed in the past (say, prior to 1990) are such that selective fields of a sniffed packet can be replaced and inserted into the network stream by a third party, without being detected.[25]

4.5 Replay

RFCs do not, in general, specify what implementers should do when they get a copy of an earlier session. SSH earlier specifications (as opposed to RFC4251-6) did not think about replay of packets, which made the earlier implementations vulnerable to password cracking.[26] The original RADIUS is also vulnerable to replay attacks.[27]

4.6 Dictionary attacks

Dictionary attacks have compromised security of protocols (e.g., Kerberos,[28] and WEP[29]) in the past. We argue that the RFCs need to specify a reasonable length for passwords as well.

4.7 Hijacking

TCP connection hijacking has long been known. When the packets are unencrypted, the attacker can make immediate productive use of the connection beyond just DoS. Encryption simply postpones this into a replay after a dictionary attack breaks it.

4.8 Man in the Middle

MiTM is similar to hijacking. Two parties, say A and B, believe they have verified each other, but M the attacker in the middle, has convinced A that he is B, and has convinced B that he is A. Communication between A and B passes through M, who typically records and alters it.

5 Ambiguous Terms

RFCs have used many ambiguous terms (e.g., “silently discard”, “tightly packed”, “left to right”, “undistinguished octets”, “reserved”, “experimental”, “is-caused-by”, “not required to honor”), but a few such as “universally unique numbers” and “unpredictable random numbers” deserve special attention.

[25]CVE-1999-0111; see also Section 3.2

[26]`www.kb.cert.org/vuls/id/565052`

[27]`www.untruth.org/~josh/security/radius/radius-auth.html,section3.4`

[28]`www.securiteam.com/tools/6V0070A6AS.html`

[29]`www.airscanner.com/pubs/wep.pdf`

5.1 Universally Unique Numbers

A number of protocols require that certain numbers (e.g., request authenticators in RADIUS RFC2865, MAC addresses in Ethernet, IP addresses in IP) they use should be (i) *universally unique numbers* and/or (ii) *temporally unique*. The terms are generally described loosely, and often imply that these two terms are equivalent. But, they are not. It turns out that it is not too difficult to rigorously define the terms [4] using discrete mathematics and logic. The essence of the difference is that (ii) refers to uniqueness in the context of what has transpired so far, whereas (i) is time independent, i.e., unique from “big bang” of the distant past to “apocalypse” of the unknown future.

The difficulty with respect to (i) and (ii) is in arriving at a practical implementation that hopes to do justice to the two obligations: (a) the “sender” should generate such a number, and (b) the “receiver” should be algorithmically able to verify that the received number is one such. Depending on whether we chose interpretation (i) or (ii), the difficulties are different.

Traditionally, the problems of (i) are “solved” by using number granting authorities and associated query/answer protocols. Because these solutions are inadequate various spoofing attacks became possible.

Regarding (ii), note that this kind of uniqueness is obviously in the context of a group of “connected” senders and receivers. Hosts A and B are “connected” not in the sense of TCP-connected, but in the sense that they did exchange a message of the protocol, or transitively there was a third party C with which they did. The problems of (ii) are solvable (if we put aside the resource requirements) as follows. Every sender and receiver has access to a global historical record of what has transpired. Using such records, (a) and (b) can be implemented. But, the resource requirements – in particular, space to store the history, and the distributed computing mechanisms to keep it up-to-date across all hosts – are immense even when the group of connected hosts is small.

5.2 Unpredictably Random Numbers

“Unpredictable” is arguably a mathematically undefinable concept. Note that to be able to speak of values in a probabilistic manner makes it predictable.

We presume that the intended meaning is that *for a third party, having observed a sequence of certain values of a field being exchanged between two parties, it is computationally infeasible to compute what the future values of exchange will be.*

The phrase "computationally infeasible" is used frequently in cryptography but is rarely defined. The general consensus on its meaning is as follows. If the time complexity of an algorithm A is a function that grows faster than any polynomial, we consider A to be computationally infeasible. A similar meaning with respect to memory (and other) resources required is included in the meaning of the phrase. On a practical level, we should understand the phrase to stand for any computation that requires either extremely long time or extremely high resource requirements even on the fastest (parallel, cluster, etc.) computer systems. Extremely long here is in the class of several (zillion?) years.

Note that the "unpredictable" property regarding a certain number in a field usually has further dependencies on other field values that are known, e.g., in the RADIUS protocol, the shared secret, the server id, and the client id that share this secret.

6 Advice to RFC Writers

Knowing how an implementer should/would understand an RFC is important to the authors of RFCs. We argue that RFCs should provide "advice" to implementers as well.

Be Aware of Silence: A protocol RFC deals with three main elements: (i) the structure of a packet, (ii) the sequences of packets, and (iii) the states of the protocol engine. The "standard" interpretation of silence is not the same for all three elements. In (i) and (ii), silence implies invalid: That is, any packet that does not satisfy the described requirements must be considered invalid. In (iii), silence implies design freedom. Where you wish to give design freedom, do so explicitly. Make an explicit sentence to the effect that any situation not covered by the RFC is an illegal one, and an RFC-compliant implementation must detect these situations and be robust. If you wish to be silent on a certain item, say so explicitly.

Be Aware of Ambiguity: If non-determinism is desired, say so explicitly. By "non-determinism" we mean the following. That there are multiple, but unambiguously described, actions that are acceptable. A single choice must be made among the alternatives. Any choice so made among these need not be consistent either.

Be Aware of Infinite Sequences: Protocol sequences can be infinitely long, but every implementation can only store a finite amount of it. Do not describe properties based on infinite sequences. "Unpredictable", "unique" are some examples of this.

Timeouts: Specify minimum and maximum time delays between certain bookmarks of the packet sequence if not between consecutive packets. Obviously, as the technology changes, so will these timeouts.

7 Advice to RFC Approval Bodies

The RFC approval process is rigorous. But, this process still does not give enough importance to security issues.

Independent Implementations: Do not approve RFCs unless at least two independent implementations that "agree" exist. Consider, e.g., the standards of publication of results in biological sciences where an experiment can take years or decades and blind studies are required.

Completeness of Packets: An approved RFC must rigorously define *valid* packets. All packets that do not satisfy this definition must be considered *invalid* packets. An approved RFC must separately state what the actions are in response to (i) valid and (ii) invalid packets.

A "good" RFC will further divide the invalid packets into (a) invalid but harmless packets, (b) invalid and resource-exhausting packets, (c) invalid and causes-hangs packets, and (d) invalid and causes-halts packets.

Completeness of States: An approved RFC must rigorously define (i) what constitutes a *state* of the protocol engine, (ii) the *valid states*, and (iii) all the rest of the states as *invalid* states. An engine must never reach a valid state consuming invalid packets. An engine must never reach an invalid state consuming valid packets. Once an invalid state is entered, the engine remains in the invalid state, transitioning into a valid state only via explicitly described packets in the RFC.

Timeouts: An approved RFC must define timeouts for every state (both valid and invalid). A timeout causes a state transition. Such transitions must be explicitly described.

8 Advice to Implementers

Even if an RFC is not written following the advice above, implementers should follow the best practices as suggested below.

Ascertain Packet Validity: Packet validity must be ascertained, *before* any action is taken. Any gain in computational efficiency obtained by ignoring this advice is doomed.

Design Freedom and Silence: Do not interpret silence as design freedom. Caution on the side of packet validity (item above) and interpret silence as invalid packets (item below).

Invalid Packet Processing: Unless the protocol explicitly describes how invalid packets should be dealt with, an implementer must "silently drop" such packets. This means that the protocol engine remains in the state that it was in prior to receiving the invalid packet. Other than a small amount of computational time, no resources are consumed. Any allocated resources are released. Logging of invalid packets is advised but must consider the requirements and security risks posed by logging.

Timeouts: Implement a timeout for practically every state of the protocol engine, except for a tiny few states explicitly identified as quiescent states.

Think About Worst-Case Scenarios: The implementer should consider the worst case scenarios before taking any action based on the contents of a packet. Some examples: (i) Make sure all resources needed will be/are available (ii) Analyze possible deadlock and live-lock scenarios and consider the corresponding prevention techniques.

9 Conclusions and Future Work

9.1 Conclusions

RFCs should be written in a way that none of the sections are ambiguous. Invariants on header values and sizes should be specified in the RFC so that the implementer would not have freedom for sloppy interpretation of packets.

It should be mandatory that the RFC writers take the protocol specification through a reasonable threat model and consider possible attack scenarios.

It should be mandatory that the RFCs propose ways to handle common attack techniques such as spoofing, flooding, replay, and reuse at all stages of the protocol.

It should be mandatory that the RFCs propose security mechanisms for providing confidentiality, integrity, authentication, authorization and non-repudiation at all stages of the protocol.

9.2 Future Work

We are actively working on solutions to make the RFCs more secure. Our work includes (i) Developing a specification language for writing RFCs, with the following design goals: (i.a) The language would give enough flexibility to the specification writer, to write unambiguous RFCs. (i.b) The language should force the implementers to validate the packets; to do all the right things for security. (i.c) The language would be easily understandable by the implementers. (ii) Supplying methods for incorporating security techniques into RFCs. (iii) Supplying methods for incorporating attack technique knowledge into RFCs. (iv) Publish an example eRFC work (see, e.g., [4]) of an existing RFC, enhancing it to include all the above mentioned functionalities.

References

[1] c0redump. UDP DoS Attack on Windows 2000 IKE. 2001. `http://www.securiteam.com/windowsntfocus/6N00G0A3FO.html`.

[2] J.Satran, K. Meth, C. Sapuntzakis, and M. Chadalapaka. RFC 3720 - Internet Small Computer Systems Interface (iSCSI). 2004. `http://www.faqs.org/rfcs/rfc3720.html`.

[3] R. Kaksonen. A functional method for assessing protocol implementation security. Technical Report 448, VTT Technical Research Centre of Finland, 2001. `http://www.inf.vtt.fi/pdf/publications/2001/P448.pdf`.

[4] P. Mateti. RADIUS Protocol Annotated in OM. Technical report, Wright State U, May 2005. 65pp., working draft.

[5] B. Meyer. On formalism in specifications. *IEEE Software*, 2(1):6–26, Jan. 1985.

[6] J. Nazario. Source code scanners for better code. 1992. `http://www.linuxjournal.com/article/5673`.

[7] V. Pothamsetty and B. Akyol. A vulnerability taxonomy for network protocols: Corresponding engineering best practice countermeasures. 2005. `www.actapress.com/PaperInfo.aspx?PaperID=17388`.

[8] G. Vigna, W. Robertson, and D. Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 21–30, New York, NY, USA, 2004. ACM Press.