Protocol Vulnerability Analysis

Sean Whalen, Matt Bishop, Sophie Engle*

May 6, 2005

Abstract

Network protocols continue to suffer from well documented vulnerabilities. Despite this, a practical methodology for classifying these vulnerabilities does not exist. In this paper, we present such a methodology.

We have developed a grammar for expressing network protocol exploits in terms of vulnerabilities and symptoms. Vulnerabilities are defined by *characteristics*, conditions which must hold for a vulnerability to exist. Symptoms are the violations of policy enabled by vulnerabilities. Exploits, then, are the pairing of vulnerabilities with their corresponding symptoms.

Using our grammar, we analyzed many protocols and present our classifications visually using syntax trees. We detail the classification process, and discuss future applications of this work towards a secure protocol design framework.

1 Introduction

A network protocol is "a specification for the format and relative timing of the messages exchanged" in a spatially distributed system [3]. The operation of a network according to a policy depends on the reliability of the protocols involved. While computer networks have become ubiquitous in our lives, protocol reliability has not improved. Instead, ubiquity has caused stagnation by preventing the re-design of broken protocols for the sake of backward compatibility.

New protocols emerge frequently, designed not only by professionals but also interns and hobbyists. Regardless of a protocol's sophistication or popularity, history has shown much protocol design to be ad hoc. At best, software engineering and hardware verification are used to increase implementation quality. Unfortunately, even the highest quality implementation implies nothing about a protocol's ability to adhere to the policy of a network.

Our work is the first step towards a new methodology of policy-adherent network protocol design. Namely, we have created a classification grammar for protocol vulnerabilities via manual analysis of extant protocols. By identifying common vulnerability characteristics, we gain insight into the fundamental problems of protocol design. In future work, we hope to apply this insight to correct existing protocols and prevent design vulnerabilities in future protocols.

*Department of Computer Science, University of California, Davis, USA, {whalen, bishop, engle}@cs.ucdavis.edu

In short, we aim to classify known protocol vulnerabilities in terms of common characteristics, and develop a “best practices” for protocol design based on these characteristics.

2 Related Work

The ultimate goal of vulnerability analysis is the creation of automated formal verification tools for both protocol specifications and implementations. However, unbounded protocol security is undecidable in theory [2, 3, 4, 5] (see Figure 1). If bounded security is decidable in practice, verification tools will likely require an integrated approach to handle reasoning at the different levels of abstraction required for such a task [6]. As such, we are first focused on a smaller task: formalizing why and how vulnerabilities occur in protocols.

	Without Nonces	With Nonces
No bounds	Undecidable	Undecidable
Infinite number of sessions, and Bounded messages	DEXPTIME	Undecidable
Finite number of sessions, choice points, and Unbounded messages	NP-Complete	NP-Complete
Finite number of sessions, and Unbounded messages	NP-Complete	NP-Complete

Figure 1: Complexity of protocol security, from [5]

This section discusses related models, formal methods, and automated tools in order to distinguish the goals of these works from our own.

2.1 Models

Previous vulnerability models include RISOS [7], PA [8], Landwehr [9], and Aslam [10]. Difficulties have been addressed in [1], citing problems with abstraction, point of view, and level of focus. Our model hopes to resolve these by taking a hierarchical approach using formal grammar.

JIGSAW [11] is a modeling language which describes attacks as “a set of capabilities that provide support for abstract attack concepts.” Their approach models attacks as opposed to vulnerabilities, and is focused more on implementation than abstract characteristics.

2.2 Formal Methods

The NRL Protocol Analyzer [12] is a Prolog-based cryptographic protocol analysis tool. The program takes as inputs a protocol specification and a description of an insecure state, and returns a reachability path if one exists. The analyzer has found flaws in several cryptographic protocols using this state space searching technique [13, 14]. However, it cannot prove the general case of protocol security or capture certain classes of replay attacks [15].

Burrows, Abadi, and Needham (BAN) logic [18] is a formalization of the authentication process. This logic takes symbolic versions of assumptions, states, and goals, and applies deductive rules to find potential flaws. BAN logic is specifically for authentication, and does not address the verification of network protocols in general. Also, [19] observes a lack of well-defined semantics and an informal "protocol idealization" step which may affect results. A formalization not specific to authentication could potentially perform protocol analysis based on our characteristics.

2.3 Automated Tools

PROTOS [16] is a Java-based automated tool for testing the robustness of protocol implementations. Protocols are defined in a custom Backus-Naur Form and fed to the application, which generates network packets with values outside the specification. This approach verifies only the syntax handling of a protocol implementation.

Property-based testing [17] takes specifications of security properties and related system calls, and modifies (*instruments*) a program's source code. The instrumented code is recompiled, executed using test cases, and validated. This execute-validate cycle is repeated until confidence is gained in correctness of the code. As with PROTOS, this approach addresses correctness at the implementation level.

3 Classification

In this section, we analyze a simple exploit with our grammar to provide future context for the reader. We also discuss the benefits of a grammar-based approach.

Consider the 802.11b wireless ethernet protocol. In a simple wireless network such as one provided by the default configuration of a commercially available access point, users may access the network by knowing the network's name (the *SSID*). Authentication is based solely on the SSID, which is transmitted unencrypted by both the access point and wireless users. An attacker can easily find this network name (and thus join the network) by sniffing wireless transmissions. Some devices offer authentication based on source MAC address, which is similarly flawed.

The protocol bases authentication on an unencrypted, easily stolen shared secret. An attacker can use this flawed authentication to access networks without permission. This violates the network administrator's policy, and so we say an exploit exists in the protocol. In this exploit, a vulnerability (the flawed authentication) creates a symptom (impersonation of an intended user by an attacker). An exploit tree is provided in Figure 2.

By structuring an exploit with a formal grammar such as BNF, we are forced to use precise terminology and unambiguous relationships. Such a well-defined system is easier to understand, and provides multiple levels of abstraction.

Our grammar is implemented as an XML DTD, allowing us to specify exploits in XML. The syntax of each exploit can be validated by any XML-compliant web browser or editor. Exploits can then be visualized as trees, allowing simple and fast interpretation of information. In addition, future work can provide greater detail at any level of the exploit tree by providing a more granular sub-grammar.

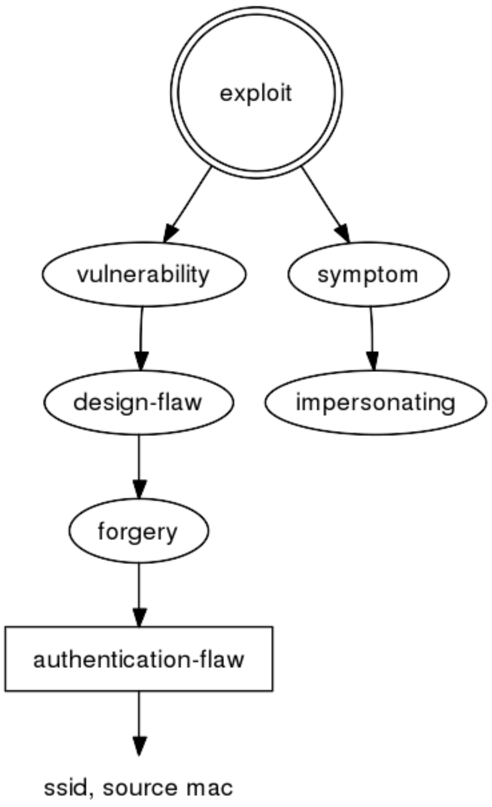

Figure 2: A simple wireless ethernet exploit, visualized as a tree

4 Definitions

Our work involves the classification of protocol vulnerabilities in terms of common characteristics. Formal classification requires precise definitions, while in computer security terms such as exploit, flaw, bug, and vulnerability are often used interchangeably. We use specific definitions of these terms, discussed in the following sections. Figure 3 visualizes our terminology relationships.

4.1 Characteristics

A condition that must hold for a vulnerability to exist is called a *characteristic*. Characteristics themselves are abstract, but are accompanied by an implementation-level description when describing real world conditions. Examples of characteristics include authentication and authorization flaws. An implementation-level description might be "no authentication in response packets". Multiple characteristics may be required simultaneously to enable a vulnerability.

A set of characteristics is *sound* if no subset of characteristics can be used to generate characteristics in the complement. In other words, the characteristics are independent. A set of characteristics is *complete* with respect to a system if its' elements can describe all vulnerabilities in that system. We hypothesize that every system has a sound, complete set of characteristics [1].

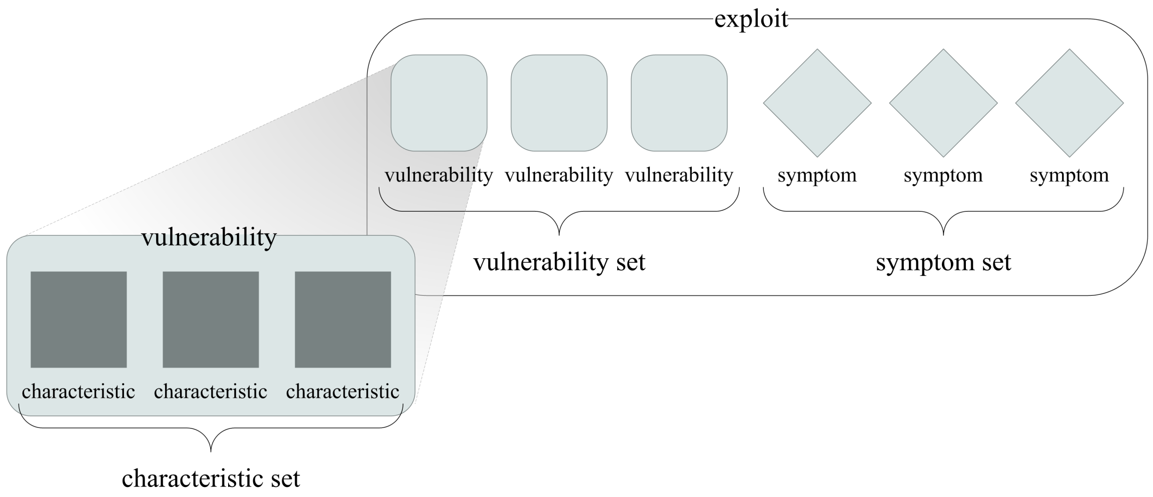

Figure 3: Terminology relationships

4.2 Vulnerabilities

A *vulnerability* is a set of conditions which enable violation of network policy. Vulnerabilities can describe multiple characteristics, and thus are more abstract. For example, a forgery vulnerability is a set containing authentication flaws, authorization flaws, and nonce flaws as characteristics.

Design vulnerabilities are inherent to a protocol specification, present even in perfect implementations. A specification using weak cryptography has a design vulnerability. In this case, the design vulnerability contains weak cryptography as a member of its' characteristic set.

In contrast, *implementation vulnerabilities* are specific to a single instance of a protocol. This could be either a hardware or software flaw. A buffer overflow in an operating system's IP stack demonstrates a software implementation vulnerability. A faulty switch port demonstrates a hardware implementation vulnerability. The overflow and the faulty port are both members of a characteristic set belonging to the implementation vulnerability.

4.3 Symptoms

A violation of network policy is called a *symptom*. Symptoms are enabled by vulnerabilities or other symptoms. Examples of symptoms include denial of service attacks, connection hijacking, and sniffing.

4.4 Exploits

Finally, an *exploit* is the pairing of one or more vulnerabilities with one or more symptoms. Vulnerabilities and symptoms are expressed as sets. Examples of exploits include ARP spoofing, TCP reset attacks, and VLAN hopping.

These definitions are key to understanding the grammar. We believe the partitioning of an exploit into vulnerabilities and symptoms greatly simplifies an otherwise convoluted analysis.

5 Grammar

5.1 Characteristics

A condition that must hold for a vulnerability to exist is called a *characteristic*. We begin by defining characteristics common to many network protocol vulnerabilities.

We are interested solely in characteristics of design vulnerabilities that are present even in ideal implementations of a protocol. However, vulnerabilities exist which require both design and implementation flaws to violate policy. Therefore, our grammar also captures implementation flaws.

We give the grammar of our characteristic set in Extended Backus-Naur Form. The set was constructed through manual analysis of many network protocol vulnerabilities. The characteristic set necessary to describe these vulnerabilities is surprisingly small. Implications of this finding are discussed in Section 6. A detailed discussion of the analysis process is found in Section 7.

Characteristic Grammar Definitions

MIS-CONFIGURATION	Device or software configuration violates network policy
MIS-SPECIFICATION	Intended semantics not captured by protocol specification, example: mandating broken encryption schemes
FLAW	An unintended condition present in a system
SOFTWARE FLAW	Application or operating system violates protocol design
HARDWARE FLAW	Hardware violates protocol design
AUTHENTICATION FLAW	Protocol verifies identities insufficiently or not at all
AUTHORIZATION FLAW	Protocol verifies permissions insufficiently or not at all
NONCE FLAW	Protocol binds data to session non-uniquely or not at all
STATE MACHINE FLAW	Protocol specification tracks internal state insufficiently or not at all, enabling flooding and brute forcing

Characteristic Grammar

```
<characteristic>  ::=  MIS-CONFIGURATION | MIS-SPECIFICATION | SOFTWARE FLAW
                       | HARDWARE FLAW | AUTHENTICATION FLAW | AUTHORIZATION FLAW
                       | NONCE FLAW | STATE MACHINE FLAW
```

5.2 Vulnerabilities

A *vulnerability* is a set of conditions which enables a violation of network policy. As with characteristics, we are interested solely in design vulnerabilities that are present even in ideal implementations of a protocol.

Vulnerability Grammar

<vulnerability>	::=	MIS-CONFIGURATION$^+$ | <implementation>$^+$ | <design>$^+$
<implementation>	::=	SOFTWARE FLAW | HARDWARE FLAW
<design>	::=	MIS-SPECIFICATION | STATE MACHINE FLAW | <forgery>
<forgery>	::=	AUTHENTICATION FLAW | AUTHORIZATION FLAW | NONCE FLAW

For example, the Network Time Protocol version 2 is exploitable because a design vulnerability exists. The vulnerability is forgery of the key index field in NTP packets, allowing an attacker to set a specific value which causes a denial of service. This particular exploit is discussed in detail in Section 7.

5.3 Symptoms

A *symptom* is a violation of policy enabled by a set of vulnerabilities. While the following grammar definitions look like characteristics, our terminology restricts characteristics to vulnerabilities. In addition, these "symptom characteristics" may not be sound or complete.

Symptom Grammar Definitions

USER CREDENTIALS	User data used to create logical host connections
USER DATA	Information created by a user
HOST CREDENTIALS	Host data used to create logical host connections
HOST DATA	Information created by a user or host
HOST SERVICE	A user or operating system process on a host
CONNECTION	Logical or physical connection for data exchange between hosts
BANDWIDTH	Channel capacity of logical or physical connection
DIVERT	To change ownership, possibly transparently
DISABLE	To make unavailable, partially or completely
SNIFFING	Acquiring broadcast or non-broadcast user data
IMPERSONATING	Using user or host credentials of others
CONNECTION HIJACKING	Taking a connection and associated credentials from their owner
MAN IN THE MIDDLE	Intercepting connections between other hosts
DENIAL OF SERVICE	Disabling a host service or connection
FLOOD	Impeding or disabling a connection by saturating bandwidth

Symptom Grammar

<symptom>	::=	DIVERT <resource> | DISABLE <resource>
<resource>	::=	<user resource> | <host resource> | <network resource>
<user resource>	::=	USER CREDENTIALS | USER DATA
<host resource>	::=	HOST CREDENTIALS | HOST DATA | HOST SERVICE
<network resource>	::=	CONNECTION | BANDWIDTH
<sniffing>	::=	DIVERT USER DATA
<impersonating>	::=	DIVERT USER CREDENTIALS | DIVERT HOST CREDENTIALS
<connection hijacking>	::=	DIVERT CONNECTION $\wedge$ DIVERT HOST CREDENTIALS
<man in the middle>	::=	<connection hijacking>$^+$
<flood>	::=	DISABLE BANDWIDTH
<denial of service>	::=	DISABLE HOST SERVICE | <man in the middle> | <flood>

5.4 Exploits

An *exploit* is the pairing of one or more vulnerabilities with one or more symptoms. Exploits are the highest level of abstraction in the grammar.

Exploit Grammar

<exploit>	::=	<vulnerability>$^+$ <symptom>$^+$

For example, the Wireless Ethernet Protocol (802.11b) contains a design vulnerability where network names and source MAC addresses are not authenticated. An attacker can use this vulnerability to impersonate other network users. This impersonation is a symptom. The combination of the vulnerability set (no authentication in name/address) and the symptom set (impersonation) is what we define as an exploit.

6 Analysis of Characteristics

6.1 The Role of Policy

When describing an exploit, the analyzer makes the implicit assumption that a symptom violates some network policy. However, policies may vary by network or subnetwork and can change over time. If a symptom does not violate a network policy, a vulnerability is *no longer exploitable in a policy sense* with respect to that network. A policy change can make irrelevant a once-exploitable vulnerability.

This is illustrated by rate-limited floods on a private LAN where overall wasted bandwidth is negligible. It may be hard to imagine a network where a distributed denial of service attack is deemed acceptable by policy, but the concept is important to capture for the flexibility of the grammar.

6.2 Algorithm

Our proposed complete characteristic set is smaller than originally anticipated: mis-specification, forgery, and statelessness are the only conditions we found necessary to describe policy violation. To explain why, we turn our attention to an abstract algorithm for detecting protocol vulnerabilities.

```
Is-Secure (Protocol $P$, Implementation $I$, Policy $\rho$)

$secure \leftarrow \neg$Is-MisConfigured $(I, \rho)$
$secure \leftarrow secure \wedge \neg$Is-MisSpecified $(P)$
$secure \leftarrow secure \wedge \neg$Has-Implementation-Flaw $(I)$
$secure \leftarrow secure \wedge \neg$Has-State-Machine-Flaw $(P)$
for each packet type $p \in P$
    $secure_p \leftarrow \neg$Has-Authentication-Flaw $(p)$
    $secure_p \leftarrow secure_p \wedge \neg$Has-Authorization-Flaw $(p)$
    $secure_p \leftarrow secure_p \wedge \neg$Has-Nonce-Flaw $(p)$
    $secure_p \leftarrow secure_p \vee$ Allows-Forgery $(p)$
    $secure \leftarrow secure \wedge secure_p$
return $secure$
```

Is-Secure first checks the configuration of a protocol by calling Is-MisConfigured. A configuration may violate one network policy and not another. Such a potential violation does not necessarily expose a design flaw, since vulnerabilities are relative to the policy.

Is-MisSpecified checks the specification against the intended semantics of the protocol. For example: a protocol may rely on a weak cryptographic primitive for encryption, such as RC4 (as demonstrated by 802.11b). A perfect vendor implementation still inherits the vulnerabilities of RC4. As such, the implementation itself is not flawed. We call this discontinuity in the specification between the intended semantics (robust encryption) and actual semantics (weak encryption) a *mis-specification.*

Has-Implementation-Flaw tests for hardware and software flaws. Their classification is left for future work.

Has-State-Machine-Flaw looks for the presence of a working state machine. Statefulness at some layer is necessary for a protocol to adhere to policies forbidding flooding or brute force symptoms. A stateless protocol must process all received packets under the assumption the sender is following protocol flow. For example, a stateless protocol would accept a disconnect acknowledgement packet from an attacker as though the victim had previously requested a disconnect from the server, when she had not. The user would then be disconnected against her wishes by the protocol.

The following lines check each packet type for authentication, authorization, and a nonce preventing replay. Flaws in these elements are the primary characteristics of true protocol vulnerabilities residing in a specification. We term this set of flaws *forgery*, which removes a protocol's ability to make decisions based on message origin, destination, or timing. Without these, a protocol is a helpless arbiter of data transfer.

Finally, policy is captured by the ALLOWS-FORGERY function. Not all packet types in a protocol may be essential according to policy. Consider a packet type for statistics, whose forgery or non-delivery has no effect on other packet types. One policy may deem these symptoms acceptable, but only for statistics packets. This algorithm allows policy flexibility at the packet level.

It is doubtful that the abstract functions of IS-SECURE can be implemented. Semantic analysis by itself is intractable due to the halting problem. The purpose of such an algorithm is to give an alternate view of the factors involved in deciding protocol security, and to show where our characteristics and definitions apply.

7 Classification (Continued)

In this section, we apply our grammar to several protocol exploits of higher complexity. We use the Dolev-Yao intruder model [20], which assumes the attacker can read, modify, and delete all network traffic and perform cryptographic operations.

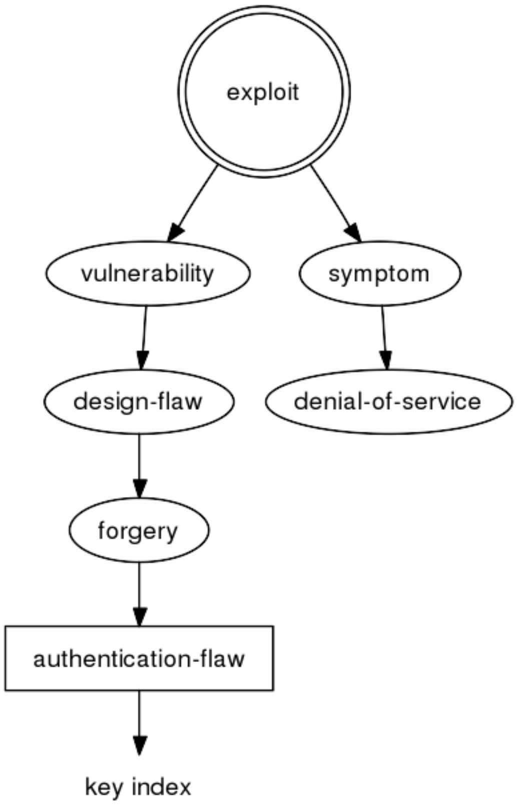

Figure 4: A network time protocol exploit

Consider the Network Time Protocol version 2. The protocol uses election and smoothing algorithms to provide robust time synchronization, preventing clock drift due to mis-behaving hosts. Optional authentication is provided by keyed-hash message authentication codes (HMAC). A field in the protocol header specifies the key used for signature verification. A key index of 0 indicates signature verification should not be performed, nor the sender trusted. This is intended by the specification for compatibility, and is not a design flaw.

However, [21] notes this key index is not included in the hashing of the message and is thus forgeable. An attacker setting the key index to 0 in-transit can thus prevent a client from participating in the election algorithm. We classify this as a denial of service, where the attacker is disabling a host's access to NTP election on a different host. Because this flaw in authentication leads to a denial of service, this is an exploit under policies forbidding such symptoms. The exploit tree is shown in Figure 4.

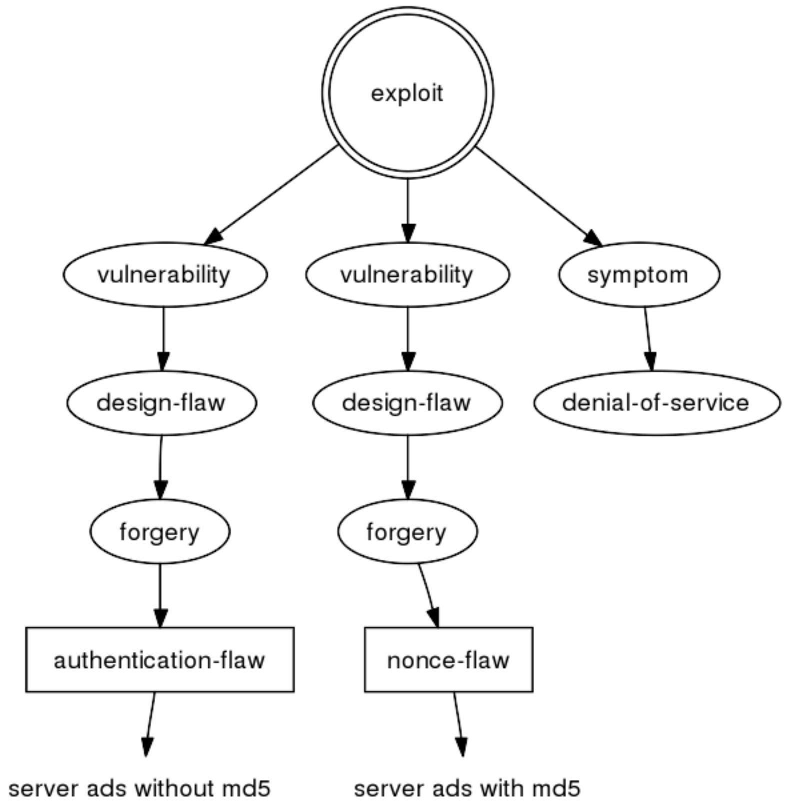

Figure 5: A VLAN trunking protocol exploit

Next, consider the Virtual LAN Trunking Protocol. The protocol automates the exchange of VLAN configuration data across a network. Switches (including those from different VLANs) can be partitioned into *domains* which confine configuration traffic.

Configuration data is sent in packets called *advertisements*. An advertisement contains a revision number, where a higher number indicates more recent data. Switches accept advertised data if the revision number is higher than their own.

Without authentication, an attacker can manipulate the VLAN topology by forging an advertisement with a high revision number. Switches can perform optional authentication based on a password hash stored in the packet. However, the lack of a nonce allows an attacker to capture the hash in transit and use it to construct a forged packet. The exploit tree is shown in Figure 5.

This is the process of classification used to develop the grammar: examine known protocol vulnerabilities, construct a set of characteristics to describe them, organize hierarchies, eliminate redundancies, and gain experimental confidence by applying the grammar to many vulnerabilities. We hope to identify new vulnerabilities in future work.

Generating statistics from 24 XML trees, we find the following: 21% contain mis-configurations, 38% contain implementation flaws, and 67% contain design flaws. Of the implementation flaws, all are software-based. Of the design flaws, 4% are mis-specifications, 21.7% are state machine flaws, 60.8% are authentication flaws, and 13% are nonce flaws.

8 Conclusion

There are many levels of abstraction in a network protocol: human semantics ("we want to hide user data from attackers"), specification semantics ("implementers must use a stream cipher for encrypting payloads"), implementation semantics (using RC4 as a stream cipher), and implementation syntax (writing flawless RC4 code which is bypassed by an input validation flaw).

To result in machine-runnable code, human concepts must cross "semantic gaps" at each of these levels. Incorrectly crossing these gaps will change the concept represented. The specification may not capture the author's intent, because unintended wording allows vendors to disable encryption. The implementation may not capture the specification's intent, because (unknown to programmers) a weak stream cipher is chosen. A combination of smaller gap errors may also alter the original concept.

The ultimate goal of vulnerability analysis, then, should be to formalize and validate the crossing of these gaps. Towards this goal, we developed a grammar to classify network protocol vulnerabilities by sets of simple conditions called *characteristics*. A small number of characteristics were needed to classify several dozen vulnerabilities. We created vulnerability trees using XML to validate our structure and allow rapid visualization.

Future work involves expansion of the grammar to include software and hardware flaws. A classification of cryptographic flaws is of key importance, since flaws in authentication comprise the majority of vulnerabilities we examined. Furthermore, a best practices based on our characteristics would inform designers not only what pitfalls to avoid, but how to avoid them.

We believe this work is important for its' ability to unambiguously classify at multiple levels of abstraction, to rapidly visualize vulernabilities, and to focus protocol designers on the small set of characteristics which continue to create vulnerabilities. With future expansion of the grammar and creation of best practices, we will have addressed both the *what* and *how* of secure protocol design.

References

[1] M. Bishop, "Vulnerability Analysis", *Proc. 2nd International Symposium on Recent Advances in Intrusion Detection*, Sep. 1999.

[2] S. Even and O. Goldreich, "On The Security of Multi-Party Ping-Pong Protocols", *Proc. 24th IEEE Symposium on Foundations of Computer Science*, 1983, pp. 34-39.

[3] L. Gong and P. Syverson, "Fail-Stop Protocols: An Approach to Designing Secure Protocols", *Proc. 5th International Working Conference on Dependable Computing for Critical Applications*, Sep. 1995, pp. 44-55.

[4] I. Cervesato et al, "A Meta-Notation for Protocol Analysis", *Proc. 1999 IEEE Computer Security Foundations Workshop*, 1999, pp. 55.

[5] M. Rusinowitch and M. Turani, "Protocol insecurity with finite number of sessions is NP-complete", *14th IEEE Computer Security Foundations Workshop*, 2001, pp. 174-190.

[6] C. Meadows, "Open Issues in Formal Methods for Cryptographic Protocol Analysis", *Lecture Notes in Computer Science*, vol. 2052, 2001, pp. 21.

[7] R.P. Abbott et al, "Security Analysis and Enhancements of Computer Operating Systems", Technical Report NBSIR 76-1041, Institute for Computer Science and Technology, National Bureau of Standards, 1976.

[8] R. Bisbey and D. Hollingworth, "Protection Analysis: Final Report ISI/SR-78-13", USC/Information Sciences Institute, May 1978.

[9] C. Landwher et al, "A taxonomy of computer program security flaws", Technical report, Naval Research Laboratory, Nov. 1993.

[10] T. Aslam, I. Krsul, and E. Spafford, "Use of a Taxonomy of Security Faults", *Proc. 19th National Information Systems Security Conference*, 1996.

[11] S. Templeton and K. Levitt, "A Requires/Provides Model for Computer Attacks", *Proc. Workshop on New Security Paradigms*, ACM, Sep. 2000.

[12] C. Meadows, "The NRL Protocol Analyzer: An Overview", *Journal of Logic Programming*, vol. 26, no. 2, 1996, pp. 113-131.

[13] C. Meadows, ""Applying Formal Methods to the Analysis of a Key Management Protocol", *Journal of Computer Security*, vol. 1, no. 1, 1992.

[14] S. Stubblebine and C. Meadows, "Formal Characterization and Automated Analysis of Known-Pair and Chosen-Text Attacks", *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, 2000, pp. 571-581.

[15] S. Malladi, J. Alves-Foss, and R. Heckendorn, "On Preventing Replay Attacks on Security Protocols", *Proc. International Conference on Security and Management*, Jun. 2002, pp. 77-83.

[16] R. Kaksonen, "A Functional Method for Assessing Protocol Implementation Security", Technical Research Centre of Finland, Jan. 2002.

[17] G. Fink and M. Bishop, "Property-Based Testing; A New Approach to Testing for Assurance", *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 4, Jul. 1997.

[18] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication", *ACM Transactions on Computer Systems*, Feb. 1990.

[19] W. Mao, “An Augmentation of BAN-Like Logics”, *Proc. 8th IEEE Computer Security Foundations Workshop*, 1995.

[20] D. Dolev and A. Yao, “On the security of public-key protocols” *Proc. IEEE Symp. on Foundations of Computer Science*, 1981, pp. 350-357.

[21] M. Bishop, “A Security Analysis of the NTP Protocol”, *6th Annual Computer Security Applications Conference*, 1990.