Security Assessment of IP-based Networks: A Holistic Approach*

Mahesh V. Tripunitara†‡, Partha Dutta† and Gene Spafford‡

CERIAS TR-99/02

Abstract

This paper deals with network security assessment. We discuss currently available network security assessment tools and provide a categorization of their limitations. We revisit the methodology that the tools are based on, the flaw-hypothesis testing methodology. We then discuss the application of the methodology to network security assessment and discuss what is necessary to augment current network security assessment tools to make the testing methodology more holistic so confidence can be placed in the results reported by the tools.

1. Introduction

Information security deals with the *confidentiality* and *integrity* of data, and *availability* of resources. A *security perimeter* delineates the scope of the security policy in effect and any security safeguards that are in place. *Attacks* are malicious acts initiated outside the security perimeter that attempt to exploit *vulnerabilities* in the system(s) within the perimeter.

A *security assessment*, in the context of this paper, is a set of controlled attacks intended to find vulnerabilities in the system. The words attack and *test* are used to mean the same in the context of a security assessment. An *IP-based network* is one that uses the TCP/IP protocol suite [1,2,3] for communication. A *network security assessment* is a security assessment of a system conducted from across a network. We use *testing* as analogous to *assessment*.

The goal of this paper is to discuss the shortcomings of current practices in network security assessment of IP-based networks and propose an approach that is more holistic. The paper provides a categorization of problems with currently used security assessment tools in discussing their limitations.

The paper then revisits the flaw-hypothesis testing methodology that such tools are based on. The paper presents architecture for network security assessment that is more effective and addresses the limitations of the currently used tools. This architecture is being used to test AT&T's Common Open IP Platform (COIPP) [4], which is an IP-based networking infrastructure for heterogeneous data.

Some of the assessment tools we studied are the ISS scanner [5], SAINT/SATAN [6] and the WebTrends scanner [7]. As our objective is not to point out flaws in specific products, we do not associate any of our examples with a specific product or software, but keep our treatment generic. We believe that the categorization of limitations in current tools that we discuss in section 3 is applicable to several, if not all, such assessment tools.

The paper is organized as follows. Section 2.1 briefly discusses the COIPP. Section 2.2 discusses how currently used network security assessment tools work. Section 3 discusses limitations with the approach and provides a categorization of problems. Section 4.1 discusses the flaw hypothesis testing methodology,

---

* Portions of this work were supported by sponsors of CERIAS. A version of this paper is to appear in the Proceedings of ISOC's INET'99, San Jose, CA, 1999.
† Internet Platforms Organization, AT&T Labs, San Jose, CA.
‡ CERIAS, Purdue University, West Lafayette, IN.

and section 4.2 discusses our enhancements based on the flaw-hypothesis testing methodology for the security assessment of IP-based networks. Concluding remarks are made in section 5.

2. Currently Used Practices for Network Security Assessment

As we mentioned in section 1, we use the COIPP from AT&T Labs [4] as the network infrastructure that is the System Under Test (SUT.) In this section, we first discuss the COIPP in section 2.1. Then, in section, 2.2, we discuss currently used network security assessment tools and practices and how they are used to test the COIPP.

2.1 The Common Open IP Platform

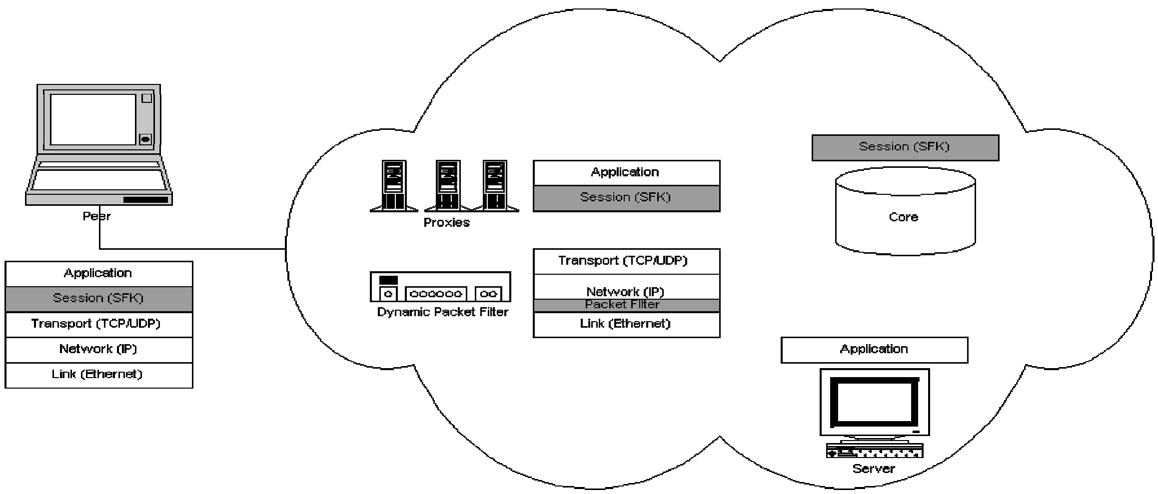

Figure 1 - The Common Open IP Platform (COIPP) and its components. The COIPP is an integrated data communications infrastructure. The figure also indicates the layers from the TCP/IP protocol suite relevant to each component in the COIPP. SFK stands for "Security Framework Kit."

Figure 1 shows the components of the COIPP that are necessary for this paper. A *gate* is a router-firewall. The firewall component includes a dynamic packet filter and proxies. The *core* contains a database that includes the security policies in effect. Each *peer* can act as either a client or a server. The COIPP provides an integrated data communications infrastructure. The components of the cloud are those that are essential to an intra-net.

The cloud is considered a *domain of trust*. That is, every component fully trusts every other component in the cloud. The cloud is the security perimeter for our tests. Peers have three different levels of trust. If a peer is not *logged in*, the cloud has the least trust in the peer. If a peer is logged in (and therefore authenticated to the cloud), the cloud trusts it to a greater extent. The highest level of trust in a peer occurs when a peer is logged in to the cloud, and a server peer clarifies that the peer is allowed access to its services.

2.2 Current Network Security Assessment Practices

Figure 2 shows how current network security assessment tools work.

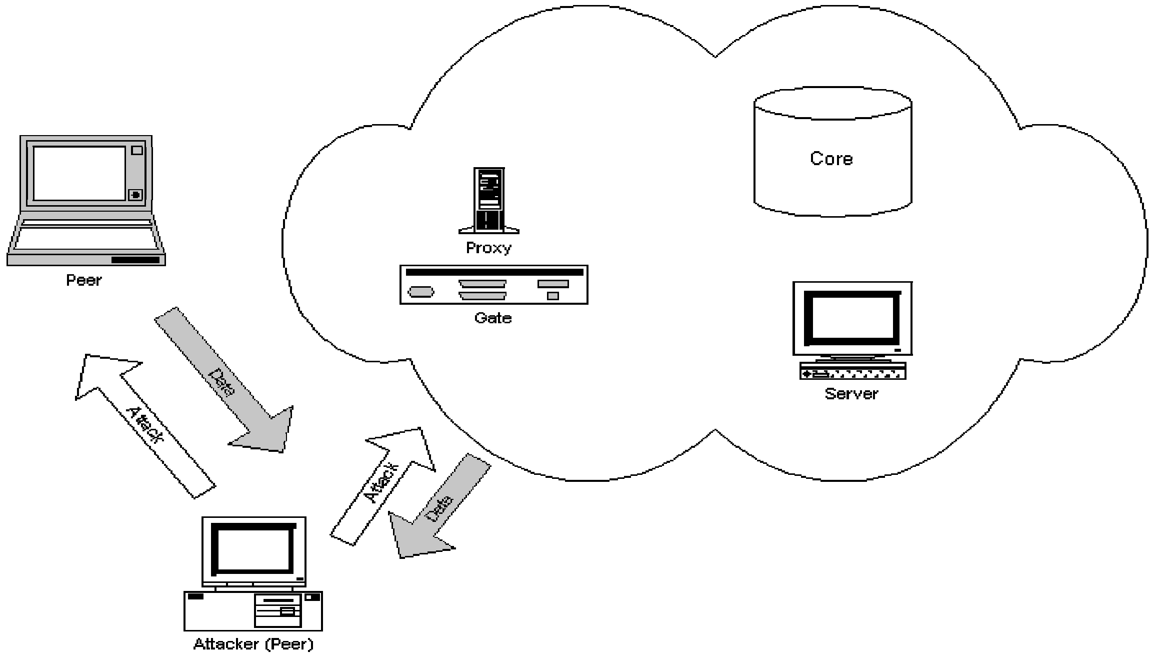

Figure 2 - How Current Network Security Assessment Tools Work in the Context of the COIPP.

As the components within the cloud are trusted, the machine that tests are generated from is a peer. The testing software is given a target IP address to test against. This IP address represents a component of the SUT that is to be tested.

The software typically begins with a step in which information on the target is collected. This involves a port scan to find out what TCP and UDP ports in the target are open for potential exploitation. The target is also checked for whether it supports protocols such as SNMP [8] and ICMP [9]. The design and implementations of some these protocols are known to have vulnerabilities.

A test consists of the testing software establishing a session with, or generating some packets intended for the target IP address and awaiting a response. The testing software decides whether a vulnerability exists based on the response.

Tests are drawn from a test database that is part of the testing software. An example of a test is to check for whether version 2 of the Post Office Protocol (POP2) [10] is running and if it is, whether clear text passwords are sent across the network every time a POP2 client establishes a connection to retrieve email. This test is done by the testing software checking whether the POP2 port is open, pretending to establish a connection and scrutinizing the data returned in establishing a connection. The initial data returned by a POP2 server indicates whether the clear text password option is the one in use, from which the testing software concludes that there is a vulnerability.

3. Limitations with Current Approaches

This section discusses limitations with the approach discussed in section 2.2 by providing a categorization of such limitations.

The problems with the results presented by current network security assessment tools are:

- False Positives: A false positive is a vulnerability that is reported to exist, but does not. As we discuss in section 4, the flaw hypothesis testing methodology that the tools are based on does not guarantee that a certain hypothesized vulnerability exists. But the test for the vulnerability needs to be constructed sufficiently well that a false positive is not reported.

The problem with false positives is that it wastes a security administrator's resources and time in analyzing whether the problem really exists or not. If complete confidence is placed in the security assessment tool, then resources could be wasted in safeguards against exploitation of the reported vulnerability that does not exist.

In the case of the security assessment of the COIPP, current tools reported several false positives. For instance, on one of the targets in the cloud, some tools reported a vulnerable POP2 server. The conclusion was incorrect, as we did not have the POP2 port open on any of the machines in the cloud.

- False Negatives: A false negative is a vulnerability that exists, but is not reported to exist by the tool, despite the tool's documentation stating that the vulnerability is checked for. A false negative can be considered more dangerous than a false positive as it gives a security administrator mistaken confidence in the security status of the system.

  The BugTraq mailing list [11], that is an open forum for discussions on security issues, recently carried a thread on false negatives in a particular security assessment product. The problem pertained to a buffer overflow in the Cisco IOS that the tool claims to check for, but did not find in a certain system.

  We refer the reader to that discussion in the BugTraq archives [11] for an example of false negatives in security assessment products.

- Inconsistent Results: Running a tool multiple times on the same system does not always produce the same reports from the tool. This, we term inconsistent results.

  For instance, during one of our tests of the COIPP, the finger service was reported to be running on one of the targets. The finger service is considered a vulnerability as it may reveal compromising information about the users on the system.

  While the finger service was indeed running, this fact was not reported when we employed the tool again. We then ran the tool several times and observed that sometimes, the tool did not report the existence of the finger service on the target.

  The danger with inconsistent results is that a security administrator can never be sure if the tool needs to be run several more times to ensure that he has found all the vulnerabilities the tool is capable of finding.

These problems arise from a combination of the limitations of the tools, which are categorized as follows:

- Software Bugs: Network security assessment tools, like any other piece of software, are buggy. A security administrator would want to be aware of such bugs and whether they impact the results presented by the tool so that he does not gain a misplaced sense of confidence in the security status of the SUT.

- Poor Data Collection: There are two issues with data collection in current network security assessment tools:
  1. Data collected is not sufficient or appropriate to conclude whether a vulnerability exists or not.
  2. Data is only collected at the location in the network that tests are generated from. This is not always sufficient to conclude whether a vulnerability exists or not.

- Poor Data Analysis: Even if sufficient data is collected to make an inference about the existence of a vulnerability, the data analysis techniques that are employed are poor, and therefore the inference is wrong.

- Insufficiency of Tests: Tests are sometimes not sufficient from two perspectives:

    1. It is possible to test for the existence of a certain vulnerability, but those tests are not carried out. A security administrator is often sold on the number of tests that a tool can perform, and not told about the vulnerabilities whose existence can be tested for, but are not.

        This is particularly the case with "low level" network vulnerabilities, in the protocol stack below the session layer (see figure 1.) Current network security assessment tools have very few tests that test for vulnerabilities in, for instance, the TCP, the IP and link (Ethernet) layers.

    2. The tool claims to test for the existence of a vulnerability, but does not carry out several possible tests for that vulnerability. An example of this is the Cisco IOS vulnerability that we mentioned earlier in this section under "false negatives."

- Incomplete Architecture: There are some vulnerabilities whose existence cannot be tested for by generating tests from one location in the network (or one peer, in the context of the COIPP.) Some of the tests may require co-ordination between peers. Current network security assessment tools typically use a single peer to generate tests from.

4 A (more) Holistic Approach to Network Security Assessment

This section describes our approach to network security assessment that augments the currently available and widely used tools. We address the issues discussed in section 3 to make the security assessment more holistic; so more confidence can be placed in the results from the assessment.

Section 4.1 revisits the flaw hypothesis testing methodology that the assessment tools are based on and discusses the problems from section 3 in the context of the methodology. Section 4.2 presents our approach to security assessment that is based on the flaw-hypothesis testing methodology.

4.1 The Flaw-Hypothesis Testing Methodology

The *flaw-hypothesis testing methodology* was first proposed in [12]. The methodology involves hypothesizing a flaw about the SUT, and then testing for whether that hypothesis is true. Note that the results from the test only indicate whether the hypothesis is true, not whether it is false.

According to [12], the methodology consists of four stages:

- Acquiring knowledge of the SUT's control structure,
- The generation of an inventory of suspected flaws,
- Confirmation of the hypotheses, and,
- Making generalizations about the underlying system weaknesses for which the flaw represents a specific instance.

The flaw-hypothesis testing methodology, when applied to security testing, is also called *penetration testing*. [13,14] discuss applying the technique to find security flaws. This methodology is appealing because it finds flaws in several aspects of a system, such as, specification, code, operation, design and policy.

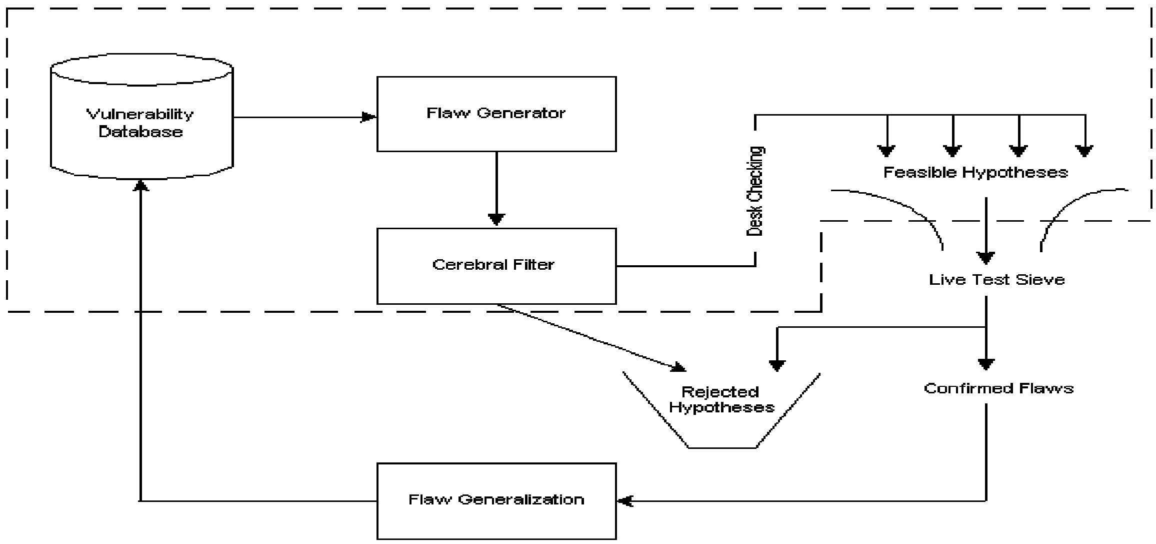

Figure 3 - The Various Stages in the Flaw Hypothesis Testing Methodology

Figure 3 reproduces figure 3-2 from [13] with some modifications. The figure indicates the four stages from above. A *vulnerability database* stores information about vulnerabilities in a manner that facilitates analysis of such vulnerabilities and the construction of tests for the existence of those vulnerabilities. One such a database is discussed in [15].

The *flaw generator* could be built into the vulnerability database. The vulnerability database includes a classification that the flaw generator can use to hypothesize flaws. Note that we make a subtle distinction between "flaw" and "vulnerability." In this context, we perceive a flaw as a specific instantiation of a vulnerability. That is, a test or attack would discover a flaw, which can then be generalized based on the vulnerability that was used to hypothesize the flaw.

The *cerebral filter* applies "common sense" and operational and policy based decisions to, perhaps, reject some hypotheses. Some of the hypotheses are only valid in certain operating environments or when certain policies are in effect.

Hypotheses deemed feasible by the cerebral filter and then tested for using the *live test sieve*. The step of mapping a hypothesized flaw to a test that can effectively test for the flaw is non-trivial. Currently, we do not have an established technique to do this and merely draw our tests from exploit scripts that are widely available and by writing our own scripts and programs.

The flaw generalization step makes inferences about the existence of a vulnerability based on the results on the tests for a flaw. This step is sometimes performed manually, especially to verify results that may be returned by current network security assessment tools.

4.2 Applying the Flaw-Hypothesis Testing to Network Security Testing

Applying the flaw-hypothesis testing methodology to testing the COIPP involves the following steps:

- Establishment of a vulnerability database: this database can be a combination of the database from the various tools that are currently available and from information sources such as CERT advisories [16]. Note that this database does not have to reside in a single location or machine.
- Establishment of an architecture to collect data from where it is deemed necessary in the network and store such data for future analysis. Note that data does not have to be collected from all points for every test.

- Establishment of a mechanism to generate and perform tests from the vulnerability database. This may involve incorporation of new tools, and tools that have not thus far been used for network security assessments. An instance of such a tool is the SPaK packet generator [17].

Currently, we are in the process of building a vulnerability database to suit the testing methodology we have employed. The database is not meant to be an analysis tool as the database discussed in [15] is, but is geared towards the hypothesizing of security flaws and the building of tests for those flaws.

For data collection, both to verify results from (existing) tools and for tests generated from those built using SPaK [17], we use tools such as snoop and tcpdump, and have shell or perl scripts that look for "interesting" information in the data collected by those tools. We locate these data collectors at every point in the network we consider affected by a test. This includes routers and switches that are intermediaries for the data transfer.

To deal with concerns about insufficient data collection mentioned in section 3, we prefer to collect "too much" data than "too little." Thus, tools such as snoop and tcpdump help, because they are raw packet sniffers.

5 Conclusions

Network security assessment was pioneered by the work on SATAN [18]. Since then, several commercial products have been released whose testing architecture is similar to that of SATAN. But these products do not always suit a security administrator's needs.

In this paper, we have discussed the shortcomings with such network security assessment tools and provided a categorization of such shortcomings. The categorization is useful, as it gives an indication of how a limitation can be addressed. We have also revisited the testing methodology that the tools are based on, the flaw-hypothesis testing methodology.

We also discussed how the flaw-hypothesis testing methodology can be used to augment existing network security assessment tools to make the testing methodology more holistic than that afforded by those tools. We have applied the enhanced testing methodology to the COIPP with success.

References

1. D.E. Comer, *Internetworking with TCP/IP*, Prentice-Hall, Englewood Cliffs, New Jersey, 3rd Edition, 1995.
2. J. Postel, *RFC-791 Internet Protocol*, Information Science Institute, University of Southern California, CA, Sept. 1981.
3. J. Postel, editor, *RFC-793 Transmission Datagram Protocol*, Information Science Institute, University of Southern California, CA, Sept. 1981.
4. Internet Platforms Organization, *21st Century Advanced Network Services Platform Technology Overview*, AT&T Labs White Paper, April 1998.
5. Internet Security Systems, *The ISS Internet Scanner*, Available from http://www.iss.net.
6. Dan Farmer and Wietse Venema, *SATAN*, Available from ftp://ftp.win.tue.nl/pub/security/.
7. WebTrends, *The WebTrends Security Analyzer*, Available from http://www.webtrends.com.
8. J. Case, M. Fedor, M. Schoffstall and C. Davin, *RFC-1098A Simple Network Management Protocol (SNMP)*, Available from http://www.faqs.org/rfcs/rfc1098.html, April 1989.
9. J. Postel, *RFC-777 Internet Control Message Protocol*, Information Science Institute, University of Southern California, CA, April 1981.
10. M. Butler, J. Postel, D. Chase, J. Goldberger and J. K. Reynolds, *RFC-937 Post Office Protocol - Version 2*, Information Sciences Institute, University of Southern California, Feb. 1985.
11. BugTraq, *About the BugTraq mailing list*, Available from http://www.geek-girl.com/bugtraq/about.html.

12. Richard R. Linde, *Operating System Penetration*, Proceedings of the National Computer Conference, Vol. 44, AFIPS Press, Montvale, N.J., 1975.
13. Clark Weissman, *Security Penetration Testing Guideline: A Chapter of the Handbook for the Computer Security Certification of Trusted Systems*, NRL Technical Memorandum 5540:082A, January 1995.
14. W. Timothy Polk, *Testing Computer System Vulnerability*, NIST SP 800-6, December 1992.
15. Ivan Krsul, *Software Vulnerability Analysis*, Ph.D. dissertation, Department of Computer Science, Purdue University, IN, 1998.
16. Computer Emergency Response Team (CERT), *CERT Advisories*, Available at http://www.cert.org/advisories/.
17. Karyl F. Stein, *SPaK - Send Packet*, Available from http://www.xenos.net/.
18. Dan Farmer and Wietse Venema, *Improving the Security of your Site by Breaking into it*, Available from http://www.alw.nih.gov/Security/Docs/admin-guide-to-cracking.101.html.
19. CERT, *Topic: Vulnerability in IMAP and POP*, CERT Advisory CA-97.09, April 7, 1997.

Appendix A

In this section, we discuss two simple examples that show problems with two current network security assessment tools. The snoop tool was used to capture raw packets for analysis.

A.1 The POP2 Buffer Overflow Problem

The POP2 buffer overflow is discussed in [19]. A tool that claimed to check for it conducted an erroneous test that resulted in a false positive. We use raw packet traces to analyze the behavior of the tool. 10.10.10.1 is the attacker's machine (that is, the machine that tests are generated from), while 10.10.10.2 is the target.

```
705   0.00000 10.10.10.1 -> 10.10.10.2 POP-2 C port=47156

           0: 0800 208f 44ab 0800 2078 eeba 0800 4500     .. .D... x....E.
          16: 002c c8be 4000 ff06 7edc 0a0a 0a01 0a0a     .,..@...~....&..
          32: 0a02 b834 006d 1331 5885 0000 0000 6002     ...4.m.1X.....`.
          48: 2238 1d66 0000 0204 05b4                    "8.f......

1174   0.00000 10.10.10.2 -> 10.10.10.1 POP-2 R port=47156

           0: 0800 2078 eeba 0800 208f 44ab 0800 4500     .. x.... .D...E.
          16: 002c 1fe2 4000 ff06 27b9 0a0a 0a02 0a0a     .,..@...'.......
          32: 0a01 006d b834 868b e900 1331 5886 6012     .&.m.4.....1X.`.
          48: 2238 adc8 0000 0204 05b4 5555       "8........UU

1175   0.00016 10.10.10.1 -> 10.10.10.2 POP-2 C port=47156

           0: 0800 208f 44ab 0800 2078 eeba 0800 4500     .. .D... x....E.
          16: 0028 c9c3 4000 ff06 7ddb 0a0a 0a01 0a0a     .(..@...}....&..
          32: 0a02 b834 006d 1331 5886 868b e901 5010     ...4.m.1X.....P.
          48: 2238 c585 0000
```

The three packets above show the TCP connection establishment phase for the POP port.

```
1176   0.00049 10.10.10.1 -> 10.10.10.2 POP-2 C port=47156 USER sssss\r\nNICK q\r\n

           0: 0800 208f 44ab 0800 2078 eeba 0800 4500     .. .D... x....E.
          16: 0042 c9c4 4000 ff06 7dc0 0a0a 0a01 0a0a     .B..@...}....&..
          32: 0a02 b834 006d 1331 5886 868b e901 5018     ...4.m.1X.....P.
          48: 2238 c69f 0000 5553 4552 2073 7373 7373     "8....USER sssss
          64: 0d0a 4e49 434b 2071 0d0a 5155 4954 0d0a     ..NICK q..QUIT..

1177   0.00061 10.10.10.2 -> 10.10.10.1 POP-2 R port=47156

           0: 0800 2078 eeba 0800 208f 44ab 0800 4500     .. x.... .D...E.
          16: 0028 1fe3 4000 ff06 27bc 0a0a 0a02 0a0a     .(..@...'.......
          32: 0a01 006d b834 868b e901 1331 58a0 5010     .&.m.4.....1X.P.
          48: 2238 c56b 0000 5555 5555 5555               "8.k..UUUUUU

1178   0.00467 10.10.10.2 -> 10.10.10.1 POP-2 R port=47156
```

```
          0: 0800 2078 eeba 0800 208f 44ab 0800 4500    .. x.... .D...E.
         16: 0028 1fe4 4000 ff06 27bb 0a0a 0a02 0a0a    .(..@...'.......
         32: 0a01 006d b834 868b e901 1331 58a0 5011    .&.m.4.....1X.P.
         48: 2238 c56a 0000 5555 5555 5555              "8.j..UUUUUU
```

The three packets above show the attacker sending some data, which contains ASCII text such as “USER sssss” followed by “NICK” and “q” and “QUIT.” The attacker then waits for a response. The response arrives followed by a FIN (connection close request) from the target.

```
1179   0.00005 10.10.10.1 -> 10.10.10.2 POP-2 C port=47156

          0: 0800 208f 44ab 0800 2078 eeba 0800 4500    .. .D... x....E.
         16: 0028 c9c5 4000 ff06 7dd9 0a0a 0a01 0a0a    .(..@...}....&..
         32: 0a02 b834 006d 1331 58a0 868b e902 5010    ...4.m.1X.....P.
         48: 2238 c56a 0000                             "8.j..

1180   0.00187 10.10.10.1 -> 10.10.10.2 POP-2 C port=47156

          0: 0800 208f 44ab 0800 2078 eeba 0800 4500    .. .D... x....E.
         16: 0028 c9c6 4000 ff06 7dd8 0a0a 0a01 0a0a    .(..@...}....&..
         32: 0a02 b834 006d 1331 58a0 868b e902 5011    ...4.m.1X.....P.
         48: 2238 c569 0000                             "8.i..

1181   0.00133 10.10.10.2 -> 10.10.10.1 POP-2 R port=47156

          0: 0800 2078 eeba 0800 208f 44ab 0800 4500    .. x.... .D...E.
         16: 0028 1fe5 4000 ff06 27ba 0a0a 0a02 0a0a    .(..@...'.......
         32: 0a01 006d b834 868b e902 1331 58a1 5010    .&.m.4.....1X.P.
         48: 2238 c569 0000 5555 5555 5555              "8.i..UUUUUU
```

The three packets above show a graceful shutdown of the connection.

The packet traces show that the tool merely checks whether the port is open for connection establishment and data exchange, and then reports that the version of POP may be vulnerable. In this case, we were running a version of POP that (based on CERT advisories [16]) is not known to be vulnerable to a buffer overflow.

A.2 The netbios-ssn Problem

One of the checks performed by some of the tools is whether the netbios session service is running. The netbios session service, if running and not properly configured, could represent a vulnerability.

This is an example of an inconsistent result reported by the tool. The tool reported the netbios-ssn as running during some of the testing instances, but not others. We again use packet traces to explain the behavior, which illustrates the need for better data collection and analysis to infer and interpret results.

During an occasion in which the tool reported the netbios-ssn as running, and therefore potentially vulnerable, the following were the TCP connection teardown packets. Again, 10.10.10.1 is the attacker and 10.10.10.2 is the target.

```
1782   0.00028 10.10.10.2 -> 10.10.10.1 TCP D=45125 S=139 Fin Ack=151785338 Seq=1946214529
Len=0 Win=8760

          0: 0800 2078 eeba 0800 208f 44ab 0800 4500    .. x.... .D...E.
         16: 0028 f634 4000 ff06 5181 0a0a 0a02 0a0a    .(ö4@...Q....i..
         32: 0a01 008b b045 7400 e081 090c 0f7a 5011    .&...Et......zP.
         48: 2238 3ba9 0000 5555 5555 5555              "8;...UUUUUU

1783   0.00011 10.10.10.1 -> 10.10.10.2 TCP D=139 S=45125     Ack=1946214530 Seq=151785338
Len=0 Win=8760

          0: 0800 208f 44ab 0800 2078 eeba 0800 4500    .. .D... x....E.
         16: 0028 72b4 4000 ff06 d501 0a0a 0a01 0a0a    .(r.@........&..
         32: 0a02 b045 008b 090c 0f7a 7400 e082 5010    .i.E.....zt...P.
```

```
         48: 2238 3ba9 0000                            "8;...

1784   0.02565 10.10.10.1 -> 10.10.10.2 TCP D=139 S=45125 Fin Ack=1946214530 Seq=151785338
Len=0 Win=8760

          0: 0800 208f 44ab 0800 2078 eeba 0800 4500    .. .D... x....E.
         16: 0028 72b5 4000 ff06 d500 0a0a 0a01 0a0a    .(r.@........&..
         32: 0a02 b045 008b 090c 0f7a 7400 e082 5011    .i.E.....zt...P.
         48: 2238 3ba8 0000                            "8;...

1785   0.00084 10.10.10.2 -> 10.10.10.1 TCP D=45125 S=139     Ack=151785339 Seq=1946214530
Len=0 Win=8760

          0: 0800 2078 eeba 0800 208f 44ab 0800 4500    .. x.... .D...E.
         16: 0028 f635 4000 ff06 5180 0a0a 0a02 0a0a    .(ö5@...Q....i..
         32: 0a01 008b b045 7400 e082 090c 0f7b 5010    .&...Et......{P.
         48: 2238 3ba8 0000 5555 5555 5555              "8;...UUUUUU
```

When netbios-ssn was not reported as running, the following were the packets corresponding to the connection teardown.

```
1804   0.00025 10.10.10.2 -> 10.10.10.1 TCP D=32966 S=139 Fin Ack=77532813 Seq=783842217
Len=0 Win=8760

          0: 0800 2078 eeba 0800 208f 44ab 0800 4500    .. x.... .D...E.
         16: 0028 1f26 4000 ff06 28b7 0a0a 0a02 0a0a    .(.&@...(....B..
         32: 0a01 008b 80c6 2eb8 7ba9 049f 0e8d 5011    .&......{.....P.
         48: 2238 1aca 0000 5555 5555 5555              "8....UUUUUU

1805   0.00015 135.197.18.38 -> 135.197.18.66 TCP D=139 S=32966     Ack=783842217
Seq=77532839 Len=0 Win=8760

          0: 0800 208f 44ab 0800 2078 eeba 0800 4500    .. .D... x....E.
         16: 0028 7c03 4000 ff06 cbd9 0a0a 0a01 0a0a    .(|.@........&..
         32: 0a02 80c6 008b 049f 0ea7 2eb8 7ba9 5010    .B..........{.P.
         48: 2238 1ab1 0000                            "8....

1806   0.00015 135.197.18.38 -> 135.197.18.66 TCP D=139 S=32966     Ack=783842218
Seq=77532839 Len=0 Win=8760

          0: 0800 208f 44ab 0800 2078 eeba 0800 4500    .. .D... x....E.
         16: 0028 7c04 4000 ff06 cbd8 0a0a 0a01 0a0a    .(|.@........&..
         32: 0a02 80c6 008b 049f 0ea7 2eb8 7baa 5010    .B..........{.P.
         48: 2238 1ab0 0000                            "8....

1807   0.00031 135.197.18.66 -> 135.197.18.38 TCP D=32966 S=139 Rst Seq=783842218 Len=0
Win=8760

          0: 0800 2078 eeba 0800 208f 44ab 0800 4500    .. x.... .D...E.
         16: 0028 1f27 4000 ff06 28b6 0a0a 0a02 0a0a    .(.'@...(....B..
         32: 0a01 008b 80c6 2eb8 7baa 0000 0000 5004    .&......{.....P.
```

ERROR: syntaxerror
OFFENDING COMMAND: ----nostringval----

STACK:

0
-0.2