19-601, Spring 2002
Homework 1

Daniel M. Vogel

January 28, 2002

1 Kerberos Authenticated Services

1.1 Kerberos Overview

Kerberos Authenticated Services (Kerberos) is a centralized trust management system. One of the key concerns with hosting multiple machines on a network, with access restrictions (authentication) concerns is trust. The goal is to allow strong authentication while minimizing the distribution of authenticating information.

In a non-Kerberized environment, to run 10 services (on different servers) with a common userlist, either each system must have independent authentication (passwords) with a userlist synchronization mechanism, or the authentication criteria (password list) must be synchronized across all machines. The first is frustrating for the user; the latter has the implication of a compromise on one machine can lead to the compromise of the whole.

Kerberos works through token distribution. A user authenticates with the Kerberos key distribution server, and is granted a token. This token, with a limited lifetime and a various service limitations (only as many priveleges as requested) is like an international travel visa – it may be used to convince the border guards of various services that their government has approved of your credentials and the appropriateness of your requests.

Naturally, the system is dependent on the security of the authentication (passwords, with various other access devices supported), the integrity, security, and availability of the key distribution server. But it is generally a very useful system (especially the flexibility of authentication forms) and is widespread. AFS, Zephyr, and most of the CMU and MIT infrastructures, for example, are dependent on it – and a variant of it has been *embraced* by Microsoft.

1.2 Vulnerabilities, CA-2000-06

CERT advisory CA-2000-06 describes buffer overflows in four kerberos subsystems: two library functions and two access control programs. The bugs in 3 of the 4 are related to support code in Kerberos version 5 to interact in Kerberos version 4 compatibility mode.

The severity of the issues are implicit in any authentication solution: to be able to grant access to different users at different access levels, the running program must have at least the highest access level rights that it can distribute. For both ksu (kerberized 'switch user' or 'super-user') and krshd (kerberized 'remote shell' daemon), this often means *root* (uid 0) access.

Therefore, if there is a bug in the authentication library that can allow a user to seize control of the client... "that's bad". krshd is especially poor because it allows a remote (potentially never authorized user) to gain access. At least the advisory hints that krshd may be a common feature on the Kerberos key distribution server – and that's **bad**, in the *whole network completely compromised* sense of the word.

1.3 MIT Patches

The MIT recommendations mostly seek to remove Kerberos 4 support.

The patches try some basic buffer tricks to make exploitation difficult. For example (from the `mit_10x_patch` patch), they avoid attempts to rewrite the command buffer (`cmdbuf`), but re-writing the "appropriate" command name. This *may* be effective, although it is terrible software engineering methodology to hardcode things like command names into sourcecode. Part of this technique:

```
--- 1469,1484 ----
          strcpy((char *) cmdbuf + offst, kprogdir);
        cp = copy + 3 + offst;

+       cmdbuf[sizeof(cmdbuf) - 1] = '\0';
        if (auth_sys == KRB5_RECVAUTH_V4) {
!         strncat(cmdbuf, "/v4rcp", sizeof(cmdbuf) - 1 - strlen(cmdbuf));
        } else {
!         strncat(cmdbuf, "/rcp", sizeof(cmdbuf) - 1 - strlen(cmdbuf));
        }
        if (stat((char *)cmdbuf + offst, &s) >= 0)
!         strncat(cmdbuf, cp, sizeof(cmdbuf) - 1 - strlen(cmdbuf));
        else
!         strncpy(cmdbuf, copy, sizeof(cmdbuf) - 1 - strlen(cmdbuf));
        free(copy);
      }
  #endif
```

As well, there is some minimistic bounds checking addition. How effective a lot of it is remains very dubious, as the following chunk of patch demonstrates (checking the size of the thing you just copied is *not* reassuring – 'how much did we just clobber?'). An example from the `mit_10x_patch` patch:

```
*** 146,151 ****
--- 146,156 ----
        }

        (void)vsprintf(p, fmt_cpy, ap);
+       /* Bounds checking??  If a system doesn't have syslog, we
+          probably can't rely on it having vsnprintf either.  Try not
+          to let a buffer overrun be exploited.  */
+       if (strlen (tbuf) >= sizeof (tbuf))
+         abort ();

        /* output the message to the local logger */
        if (send(LogFile, tbuf, cnt = strlen(tbuf), 0) >= 0 ||
```

In addition to the MIT Patches, the CERT advisory also demonstrated how widespread and diverse the commonly available kerberos codebases are (MIT Kerberos, KTH, along with NetBSD's crypto policies, etc).

1.4 Future Concern?

The advisory, and the nature of the bugs discovered, reiterate some the problems of Kerberos:

- Compatibility (Backwards, Forwards)
  implications of bugs in old versions being manifested, or in this case, bugs introduced in compatibility with the old
- Heavily Distributed, Heavily Patched
  many of the large-infrastructure environments, of which kerberos was developed for and is ideal in, take great pride in modifying and tweaking. CMU and MIT are both primary guilty parties. in such a case, patching blindly won't work; bugs may be introduced in the hacking; etc.

- Quickly Constructed, Heavily Patched
  as seen above, the software design methodolgy and software engineering approach are not positive indicators that good security coding technique is a priority.

- Critical
  networks with kerberos are dependent of kerberos. they are also very tempting targets, and a compromise in kerberos is (potential) compromise of the whole system.

Given the size and development methodology of kerberos, it would not be improbable that there will be additional security concerns in the future. The protocols (v4, v5) are fairly well analysized, so that is less likely. The code is available, and so these *should* be less likely; but like all large code bases, the problems are more likely to be manifested in the corner, support code. The active code is scrutinized in each release.

2 RealPlayer Buffer Overflow [SCN #01]

As recently reported to the BugTraq mailing list, there is a Buffer Overflow in RealPlayer 8 (and earlier).

`Date`: Thu, 24 Jan 2002 19:17:41 -0800
`From`: tmorgan-security@kavi.com
`To`: bugtraq@securityfocus.com
`Subject`: RealPlayer Buffer Overflow [Sentinel Chicken Networks Security Advisory #01]

"The problem boils down to the fact that RealPlayer trusts the format of input from real media files a bit too much." This advisory was published once the author was notified that the vendor was issuing patches (within a week of notification!), and published before there were known exploits.

The Buffer Overflow was identified through corruption of the header information on RealAudio media files. Through shifting two bytes on a stock encoded file, the playing software became unstable (corruption in the heap, with some 'copy to stack' issues). This problem probably manifested itself because Real is the sole provider of Real media encoding functionality – so as long as a file was generated by their stuff, everything was fine.

RealPlayer is a very wide-spread client program, on all major client platforms (Mac, Windows, Linux, Solaris). The potential consequences, assuming an exploit were developed:

- Worm-Trojan behavior?
  Rather than rely on a macro-system (VBS) and renaming files to look *like a different file format* (`.doc.pif`)*, this would actually* be *a common format to email*

- Higher Website Compromise implication
  If a media-rich website were compromised, and without detection/ defacing, this attack could be used in a trojan form

With as many copies as are deployed in the field, this will still be a risk in the months to come. However, the complexity of the potential exploit (smashing the heap, cross-platform systems) and the ease of detection may keep this an unrealized problem.

`http://www.sentinelchicken.com/advisories/realplayer/`

3 Fixing Buffer Overflows

3.1 Personal Responsibility

If, and when, I have been responsible for the integrity, confidentiality, and overall security of information, there are several rules of thumb to follow. Buffer Overflows are not inevitable in responsible development – there are development techniques, methodologies, and technology that will avoid them. They are often introduced not through "laziness", but with alternative objectives beyond security. Most often, this is feature-enriching.

Therefore, when selecting code to use, I try and satisfy these criteria:

- Source Code available.
  For so many reasons (security through obscurity that isn't, personal and peer inspections, customizing, cost). Or, more specifically, there are several disadvantages *and no real* advantages *to using closed-source packages.*
- Only use code written by people more concerned with security than myself.
  (*This list includes Theo de Raadt* [OpenBSD] *and D.J. Bernstein* [qmail, daemon tools, djbdns])
- Only allow services which I understand, and understand the security, authentication, and accountability implications of, at least at a high-level way.
- Development track records matter.
- Recognize the value and fragility of the information being protected, and design security policy which is reasonable in context.
  Basically, if the information is mostly valuable for its availability, concentrating on reliable backups and data integrity is better than getting hung up on advanced authentication or ubiquitous end-to-end encryption; if the confidentiality of the data is more important than accessibility, this guides policy in a different direction. There is only so much that can or should be done in security, and maximizing relevancy is the ideal.

Using these, the risk exposure and consequences of Buffer Overflows (and a host of other issues) is reduced.

3.2 Government Difference

I certainly believe the government *could* make a difference, but I'm not sure it would be positive or would not have more severe implications. In general, the most constructive difference the government could make would be through its economic, not regulatory, clout – the US government is a very large customer of information systems. By imposing, throughout the system, restrictions similar to the guidelines above – and/or structuring contracts with closed-source vendors that actually hold them responsible for false security claims with severe penalties, I believe there would be positive improvements.