附录F

摘要认证

本附录包含了实现 HTTP 摘要认证功能所需的支撑数据和源代码。

F.1 摘要WWW-Authenticate指令

表 F-1 根据 RFC 2617 中的描述，对 WWW-Authenticate 指令进行了说明。与往常一样，最新细节请参见官方规范。

表F-1 （来自RFC 2617的）摘要WWW-Authenticate首部指令

指　令	描　述
realm	显示给用户的字符串，这样用户就可以知道该使用哪个用户名和密码了。这个字符串中至少应该包含执行认证功能的主机名字，此外可能还会说明可能拥有访问权的用户的集合。例如，registered_users@gotham.news.com
nonce	服务器特有的数据字符串，每次产生一个 401 响应时都应该生成一个唯一的数据字符串。建议这个字符串为 Base-64 或十六进制数据。需要特别说明的是，由于此字符串是放在首部行中作为引用字符串传送的，所以不允许使用双引号。 nonce 的内容是与实现有关的。实现的质量取决于选择是否合适。比如，可以将 nonce 构造成以下内容的 Base-64 编码： `time-stamp H(time-stamp ":" ETag ":" private-key)` 其中 time-stamp 是服务器生成的时间或其他不重复的数值，ETag 是与所请求实体有关的 HTTP ETag 首部的值，private-key 是只有服务器知道的数据。使用这种形式的 nonce，服务器会在收到客户端的 Authentication 首部之后重新对散列部分进行计算，如果与该首部的 nonce 不符，或者时间戳的值不够近，就可以拒绝请求。通过这种方式，服务器可以限制 nonce 的有效时间。包含 ETag 可以防止对资源更新版本的重放请求。（注意：在 nonce 中包含客户端的 IP 地址，看起来好像为服务器提供了限制最初获得此 nonce 的客户端重用 nonce 的能力，但这样会破坏代理集群，来自单个用户的请求通常都会经过集群中不同的代理进行传输。而且，IP 地址欺骗也不是很难。） 实现可以选择不接受以前用过的 nonce，或以前用过的摘要，以防止重放攻击，或者选择为 POST 或 PUT 请求使用一次性 nonce 或摘要，为 GET 请求使用时间戳
domain	一个引用的、由空格分隔的 URI 列表（如 RFC 2396，“Uniform Resource Identifiers: GenericSyntax”所述），这些 URI 定义了保护空间。如果 URI 是个 abs_path，它就是相对于受访服务器的典型根 URL 的。这个列表中的绝对 URI 所指的服务器可能不是受访服务器。 客户端可以用这个列表来判定应该将同样的认证信息发送给哪个 URI 集：可以假定所有以此列表中的 URI 作为前缀的 URI（在将两者都转换为绝对 URI 之后）都位于同一个保护空间内。 如果省略了这条指令，或者其值为空，客户端就应该假定保护空间中包含了响应服务器上的所有 URI。 这条指令在 Proxy-Authenticate 首部是无意义的，此时，保护空间总是包括整个代理；如果提供了这条指令，也应该将其忽略

574

（续）

指　令	描　述
opaque	一个由服务器指定的数据串，应该由客户端不经修改地放在后继请求的 Authorization 首部中返回，这些后继请求应使用同一保护空间内的 URI。建议这个字符串采用 Base-64 或十六进制的数据
stale	一个标志，用来说明由于 nonce 值太过陈旧，前一条来自客户端的请求被拒绝了。如果 stale 为 TRUE（不区分大小写），客户端可能希望以新加密的响应重试请求，而不用再次提示用户输入新的用户名和密码。只有在服务器收到一条 nonce 无效，但摘要有效的请求（说明客户端知道正确的用户名 / 密码）时，才应该将 stale 设置为 TRUE。如果 stale 为 FALSE，或者除 TRUE 之外的其他值，或者没有提供 stale 指令，用户名和 / 或密码就是无效的，需要获取新的值

algorithm	一个字符串，说明了一对儿用来生成摘要和校验码的算法。如果没有提供这个字符串，就假定它为“MD5”。如果不识别此算法，就忽略这种质询（如果有多个算法的话，就使用另外一个）。
	在这份文档中，用“KD(secret,data)”来表示用密码“secret”对数据“data”使用摘要算法得到的字符串，而对数据“data”使用校验和算法得到的字符串则表示为“H(data)”。表示法“unq(X)”表示引用字符串“X”的值（不包含左右两边的引号）。
	对 MD5 和 MD5-sess 算法来说：
	`H(data) = MD5(data)`
	`HD(secret, data) = H(concat(secret, ":", data))`
	也就是说，摘要就是将密码的 MD5 与冒号和数据连接在一起。MD5-sess 算法目的是支持使用高效的第三方认证服务器

qop	这条指令是可选的，只是为了与 RFC 2069[6] 后向兼容才保留的。所有与此版本的摘要方案兼容的实现都应该使用它。 如果提供了这条指令，它就是由一个或多个标记构成的引用字符串，用来说明服务器所支持的“安全保障”值。值 auth 说明要进行认证，值 auth-int 说明要进行具有完整性保护的认证。一定要忽略那些不识别的选项
<extension>	未来可以通过这条指令进行扩展。要忽略所有不认识的指令

F.2　摘要Authorization指令

表 F-2 根据 RFC 2617 的描述，对每条摘要 Authorization 指令都进行了说明。最新的细节请参见官方规范。

575

表F-2　（来自RFC 2617的）摘要Authorization首部指令

指　令	描　述
username	指定域中的用户名
realm	在 WWW-Authenticate 首部中传送给客户端的域

（续）

指　令	描　述
nonce	在 WWW-Authenticate 首部中传送给客户端的那个与服务器相同的 nonce
uri	来自请求行请求 URI 中的 URI。由于代理可以在传输中修改请求行，所以会出现重复。进行正确的摘要验证计算可能需要原始 URI
response	这个就是实际的摘要——摘要认证的重点！响应是一个由 32 个十六进制数字组成的字符串，由沟通好的摘要算法生成，用来证明用户知道这个密码
algorithm	一个字符串，说明了用来生成摘要和校验和的一对儿算法。如果未提供，就将其假定为 MD5
opaque	服务器在 WWW-Authenticate 首部指定的数据串，应该由客户端不经修改地在后继请求的 WWW-Authenticate 首部中返回，这个请求应使用同一保护空间内的 URI
cnonce	如果发送了 qop 指令，就一定要使用这条指令，如果服务器没有在 WWW-Authenticate 首部字段中发送 qop 指令，就一定不能使用这条指令。 cnonce 值是客户端提供的不透明引用字符串值，客户端和服务器都用它来避免选择明文攻击，以提供双向认证以及一些报文一致性检查。 参见本附录稍后介绍的响应摘要和请求摘要计算
qop	说明客户端对报文应用的“安全保障”是什么。如果提供了这条指令，它的值就必须是服务器在 WWW-Authenticate 首部中说明的、它支持的可选值之一。这些值会影响请求摘要的计算方式。 它只是个单独的标记，而不是 WWW-Authenticate 中那样的可选值引用列表。这条指令是可选的，以保持对 RFC 2069 最小实现的后向兼容，但如果服务器说明它是通过在 WWW-Authenticate 首部字段中提供 qop 指令来支持 qop 的，就应该使用这条指令
nc	如果发送了 qop 指令，就一定要指定这条指令，如果服务器没有在 WWW-Authenticate 首部字段中发送 qop 指令，就一定不能使用这条指令。 其值是个十六进制值，表示客户端已经发送的包含 nounce 值的请求次数（包括当前请求在内）。比如，作为对指定 nonce 值的响应发送的第一条请求中，客户端会发送 nc="00000001"。 这条指令的目的是允许服务器通过维护自己保存的此计数值副本来发现请求重放——如果看到了两次相同的 nc 值，就说明请求是重放的
<extension>	未来可以通过这条指令进行扩展。所有不认识的指令都要忽略掉

F.3　摘要Authentication-Info指令

表 F-3 根据 RFC 2617 的描述，对每条 Authentication-Info 指令都进行了说明。
576 最新的细节请参见官方规范。

表F-3 （来自RFC 2617的）摘要Authentication-Info首部指令

指　令	描　述
nextnonce	nextnonce 指令的值是服务器希望客户端为未来的认证响应使用的 nonce。服务器可能会发送带有 nextnonce 字段的 Authentication-Info 首部，作为实现一次性 nonce 或修改 nonce 的手段。如果提供了 nextnonce 字段，客户端在为下一条请求构建 Authorization 首部时就应该使用它。客户端如果没能做到，就会收到来自服务器的 "stale=TRUE" 认证请求。 服务器实现应该仔细地考虑使用这种机制带来的性能影响。如果每条响应都包含了必须在服务器接收的下一条请求中使用的 nextnounce 指令，就不可能使用管道化请求了。应该考虑在性能和安全之间进行一些平衡，允许在有限的时间内使用老的 nonce 值，以实现请求的管道化。使用 nounce 计数可以在不影响管道化的情况下，维护一个新的服务器 nonce 的大部分安全优势
qop	说明了服务器应用到响应上的“安全保障”选项。值 auth 说明要进行认证，值 auth-int 说明要进行带有完整性保护的认证。服务器在响应中使用的 qop 指令值应该与客户端在相应请求中发送的值相同

rspauth	response auth 指令中的可选响应摘要支持双向认证——服务器证明了它知道用户的密码，而且通过 qop="auth-int"，它还为响应提供了有限的完整性保护。除了当 qop="auth" 或者没有在 Authorization 首部为请求指定 qop 的情况，response-digest 值的计算方式与 Authorization 首部的 request-digest 类似，A2 为：
	`A2 = ":" digest-uri-value`
	当 qop="auth-int" 时，A2 为：
	`A2 = ":" digest-uri-value ":" H(entity-body)`
	其中 digest-uri-value 是请求的 Authorization 首部中 uri 指令的值。cnonce 和 nc 值一定要与此报文所响应的客户端请求中的相应值相同。如果指定了 qop="auth " 或者 qop="auth-int"，就必须提供 rspauth 指令

cnonce	cnonce 值一定要与此报文所响应的客户端请求中的相应值一样。如果指定了 qop="auth " 或 qop="auth-int"，就必须提供 cnonce 指令
nc	nc 值一定要与此报文所响应的客户端请求中的相应值一样。如果指定了 qop="auth " 或 qop="auth-int"，就必须提供 nc 指令
<extension>	未来可以通过这条指令进行扩展。所有不识别的指令都要忽略掉

F.4 参考代码

下列代码实现了 RFC 2617 中 H(A1)、H(A2)、request-digest 和 response-digest 的计算。它使用了 RFC 1321 中的 MD5 实现。

F.4.1 文件digcalc.h

```
#define HASHLEN 16
typedef char HASH[HASHLEN];
#define HASHHEXLEN 32
typedef char HASHHEX[HASHHEXLEN+1];
#define IN
#define OUT
/* calculate H(A1) as per HTTP Digest spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    );

/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1, /* H(A1) */
    IN char * pszNonce, /* nonce from server */
    IN char * pszNonceCount, /* 8 hex digits */
    IN char * pszCNonce, /* client nonce */
    IN char * pszQop, /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod, /* method from the request */
    IN char * pszDigestUri, /* requested URL */
    IN HASHHEX HEntity, /* H(entity body) if qop="auth-int" */
    OUT HASHHEX Response /* request-digest or response-digest */
    );
```

577

F.4.2 文件 "digcalc.c"

```
#include <global.h>
#include <md5.h>
#include <string.h>
#include "digcalc.h"

void CvtHex(
    IN HASH Bin,
    OUT HASHHEX Hex
    )
    {

unsigned short i;
unsigned char j;
for (i = 0; i < HASHLEN; i++) {
    j = (Bin[i] >> 4) & 0xf;
    if (j <= 9)
        Hex[i*2] = (j + '0');
    else
        Hex[i*2] = (j + 'a' - 10);
    j = Bin[i] & 0xf;
```

```
    if (j <= 9)
        Hex[i*2+1] = (j + '0');
    else
        Hex[i*2+1] = (j + 'a' - 10);
    };
    Hex[HASHHEXLEN] = '\0';
    };

/* calculate H(A1) as per spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    )
{
    MD5_CTX Md5Ctx;
    HASH HA1;
    MD5Init(&Md5Ctx);
    MD5Update(&Md5Ctx, pszUserName, strlen(pszUserName));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszRealm, strlen(pszRealm));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszPassword, strlen(pszPassword));
    MD5Final(HA1, &Md5Ctx);
    if (stricmp(pszAlg, "md5-sess") == 0) {
        MD5Init(&Md5Ctx);
        MD5Update(&Md5Ctx, HA1, HASHLEN);
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
        MD5Final(HA1, &Md5Ctx);
    };
    CvtHex(HA1, SessionKey);
};
/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1, /* H(A1) */
    IN char * pszNonce, /* nonce from server */
    IN char * pszNonceCount, /* 8 hex digits */
    IN char * pszCNonce, /* client nonce */
    IN char * pszQop, /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod, /* method from the request */
    IN char * pszDigestUri, /* requested URL */
    IN HASHHEX HEntity, /* H(entity body) if qop= "auth-int" */
    OUT HASHHEX Response /* request-digest or response-digest */
    )
{
    MD5_CTX Md5Ctx;
    HASH HA2;
    HASH RespHash;
    HASHHEX HA2Hex;
    // calculate H(A2)
```

578

```
    MD5Init(&Md5Ctx);
    MD5Update(&Md5Ctx, pszMethod, strlen(pszMethod));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszDigestUri, strlen(pszDigestUri));
    if (stricmp(pszQop, "auth-int") == 0) {
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, HEntity, HASHHEXLEN);
};
MD5Final(HA2, &Md5Ctx);
CvtHex(HA2, HA2Hex);
// calculate response
MD5Init(&Md5Ctx);
MD5Update(&Md5Ctx, HA1, HASHHEXLEN);
MD5Update(&Md5Ctx, ":", 1);
MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
MD5Update(&Md5Ctx, ":", 1);
if (*pszQop) {
       MD5Update(&Md5Ctx, pszNonceCount, strlen(pszNonceCount));
       MD5Update(&Md5Ctx, ":", 1);
       MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
       MD5Update(&Md5Ctx, ":", 1);
       MD5Update(&Md5Ctx, pszQop, strlen(pszQop));
       MD5Update(&Md5Ctx, ":", 1);
    };
    MD5Update(&Md5Ctx, HA2Hex, HASHHEXLEN);
    MD5Final(RespHash, &Md5Ctx);
    CvtHex(RespHash, Response);
};
```

F.4.3 文件digtest.c

```
#include <stdio.h>
#include "digcalc.h"

void main(int argc, char ** argv) {
    char * pszNonce = "dcd98b7102dd2f0e8b11d0f600bfb0c093";
    char * pszCNonce = "0a4f113b";
    char * pszUser = "Mufasa";
    char * pszRealm = "testrealm@host.com";
    char * pszPass = "Circle Of Life";
    char * pszAlg = "md5";
    char szNonceCount[9] = "00000001";
    char * pszMethod = "GET";
    char * pszQop = "auth";
    char * pszURI = "/dir/index.html";
    HASHHEX HA1;
    HASHHEX HA2 = "";
    HASHHEX Response;
    DigestCalcHA1(pszAlg, pszUser, pszRealm, pszPass,
        pszNonce, pszCNonce, HA1);
        DigestCalcResponse(HA1, pszNonce, szNonceCount, pszCNonce, pszQop,
        pszMethod, pszURI, HA2, Response);
    printf("Response = %s\n", Response);
};
```

579 ~ 580