



# リバースエンジニアリングと セキュリティ脆弱性分析

Fourteenforty Research Institute, Inc.  
株式会社 フォティーンフォーティ技術研究所  
<http://www.fourteenforty.jp>

取締役副社長 最高技術責任者  
鵜飼裕司



# mov [chapter],01h

```

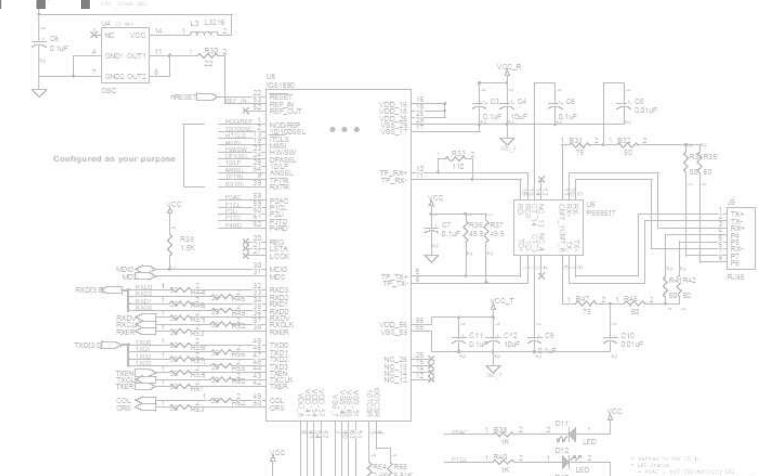
int packet_analysis(GDDCONFIG *gddc,unsigned char *packet,unsigned long length)
{
  struct ip_header /* IP header */
  struct tcp_header /* TCP header */
  char tcp_data /* TCP data */
  struct in_addr addr; /* IP address */
  char sourceIP[16]; /* Source IP address */
  char destIP[16]; /* Destination IP address */
  unsigned short sourcePort; /* Source Port */
  unsigned short destPort; /* Destination Port */
  unsigned long len_data; /* Length of data part */
  unsigned long iph_len; /* Length of IP header */
  unsigned long tcph_len; /* Length of TCP header */
  unsigned long sequence; /* Expected sequence */
  int portindex; /* Index number of port list */
  int direction; /* Direction */
  unsigned char logtype; /* Log type */
  CONN_LIST conn_list; /* Connection table list */
  static char time_t;
  struct timeval t;
  char *timep=NULL;
  char *timesp=NULL;
  char *c;

  /* Get the length of IP header and check length of IP */
  if (sizeof(struct ethhdr) < MIN_SIZE_IP+MIN_SIZE_TCP) return(0);
  ip_header = (struct ip *) (packet+sizeof(struct ethhdr));
  if (ip_header->ip_v == 4) return(0);
  if (ip_header->ip_tos > 0) return(0);
  if (ip_header->ip_len < MIN_SIZE_IP+MIN_SIZE_TCP) return(0);
  if (ip_header->ip_len > length-sizeof(struct ethhdr)) return(0);
  if (ip_header->ip_len > length-sizeof(struct ethhdr)) return(0);

  /* Get the length of TCP header and check length of TCP */
  tcp_header = (struct tcp_header *) (ip_header+iph_len);
  tcph_len = (unsigned long) (tcp_header->th_off)*4;
  if (tcph_len < MIN_SIZE_TCP) return(0);
  if (tcph_len > length-iph_len) return(0);

  /* Get the source and destination IP address */
  memcpy(&sourceIP, ip_header->ip_src.s_addr, 16);
  memcpy(&destIP, ip_header->ip_dst.s_addr, 16);
  if (!strcmp(sourceIP, destIP)) return(0);
}

```



00001B70	FF 15 F0 11	W0 01 E9	CC 03 00 00 E8	00 00 00 00	3 離...
00001B80	33 F6 56 00	BF FC FF	EE 65 18 00	80 03 00 00	Vj...51...5...
00001B90	56 6A 02 FF	35 6C 80	00 01 EF	35 D0 57 00 01	7...u.h...5...
00001BA0	15 CC 11 00	01 85 C0	75 1D 68	10 10 00 00	P...5D...62...
00001BB0	50 80 00 01	FF 35 44	80 00 01	FF 35 D0 87 00	...62...
00001BC0	FF 15 04 12	00 01 FF	35 D0 87 00	01 FF 15 2C 12	...5P...5...
00001BD0	00 01 FF 35	D4 88 00 01	FF 15 58 10	00 01 E9 64	...3.9...
00001BE0	03 00 00 83	FE 1A 77	47 0F 84 59	03 00 83 FE	...二...5...
00001BF0	11 0F 85 16	01 00 00	33 F6 39 35	E8 87 00 01	...OP...5...
00001C00	22 88 3D 28	12 00 01	56 FF D7	56 FF D7 68	...h...h...5...
00001C10	00 00 FF 35	50 80 00 01	FF 35 80 00	00 01 E9 7D	...子...5...驗...
00001C20	02 00 00 01	00 00 00	00 00 00	00 00 00 00	...OP...5...驗...
00001C30	7D 4 88 00	00 00 39	F0 F 8	80 00 00 35	...OP...5...驗...
00001C40	F0 F 84 00	00 00 83	F 1C 0	85 1 00 00	...OP...5...驗...
00001C50	00 00 83	10 80 00	A1 E8	07 01 80 00	...OP...5...驗...
00001C60	00 01 38	67 08 38	CE 0F 84	D9 02 00 00	...OP...5...驗...
00001C70	14 12 00 01	51 50 68	B1 00 00 00	FF 35 D4 87 00	...OP...5...驗...
00001C80	01 E9 56	01 00 00	8B 3D 14	12 00 01 68	...OP...5...驗...
00001C90	01 68 EC	87 00 01	68 80 00 00	FF 35 D4 87 00	...OP...5...驗...
00001CA0	01 FF D7	A1 EC 87 00	01 88 00 00	F0 87 00 01 3B	...OP...5...驗...
00001CB0	75 11 89	35 EC 87 00	01 89 35	F0 87 00 01 E9	...OP...5...驗...
00001CC0	02 00 00 51	50 E9 07 01	00 00 88	CE B8 12 01 00	...OP...5...驗...
00001CD0	00 2B C8	0F 84 3C 02	00 00 83	E9 04 0F 84	...OP...5...驗...
00001CE0	00 00 49	0F 84 F9 01	00 00 81	E9 1C 01 00 00	...OP...5...驗...
00001CF0	84 E0 01 00	00 81 E9	56 00 00 00	0F 84 3D 01 00	...OP...5...驗...
00001D00	00 81 E9	58 7C 00 00	0F 84 02 01	00 00 3B 35 5C	...OP...5...驗...
00001D10	88 00 01 0F	85 EE 00 00 00	85 45 14	8B 48 0C 8B	...OP...5...驗...
00001D20	C1 8B D1	F7 D0 C1 EA	02 83 E0 01	83 E2 01 F6 C1	...OP...5...驗...

# call [Introduction]



## はじめに

- 近年、攻撃者側の脆弱性発見・分析スキルのプロ化が進んでいる
  - 0-day脆弱性がいきなり実攻撃に利用させるケースが多発
  - Exploitの高度化
- 近年のメジャーアプリケーションの多くは、一定のセキュリティレベルを満たしている
  - ブラックボックスアプローチの脆弱性検査は非効率
- しかし、その設計・実装にはまだまだ脆弱性が多い
  - リバースエンジニアリングによる検査が効率的
  - 攻撃者側はリバースエンジニアリングを駆使して脆弱性を発見
  - 攻撃者側の脆弱性発見技術はここ数年で飛躍的に向上







## 本講演で取り上げる内容

- 近年の脅威と攻撃者が注目する脆弱性のトレンド
- リバースエンジニアリングによる効率的な脆弱性発見のポイント
- 近年のファイルフォーマット系脆弱性と発見のコツ
- ファイルフォーマット系脆弱性の実例と発見のアプローチ





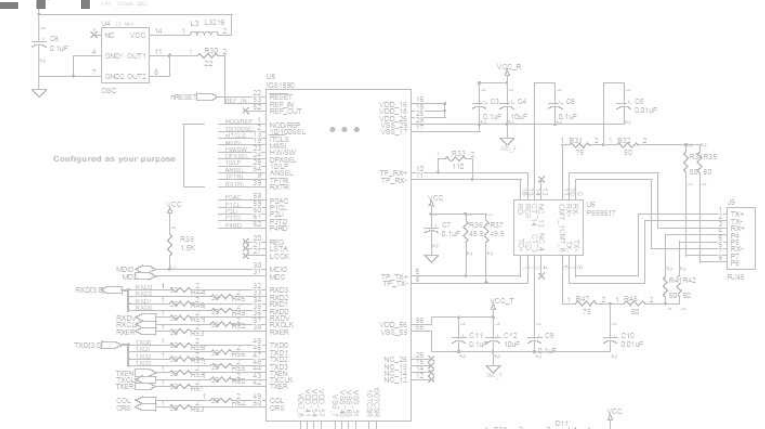
# mov [chapter],02h

```

int packet_analysis(GDDCONFIG *gddc,unsigned char *packet,unsigned long length)
{
  struct ip_header /* IP header */
  struct tcp_header /* TCP header */
  char *tcp_data; /* TCP data */
  struct in_addr addr; /* IP address */
  char sourceIP[16]; /* Source IP address */
  char destIP[16]; /* Destination IP address */
  unsigned short sourcePort; /* Source Port */
  unsigned short destPort; /* Destination Port */
  unsigned long len_data; /* Length of data part */
  unsigned long iph_len; /* Length of IP header */
  unsigned long tcph_len; /* Length of TCP header */
  unsigned long sequence; /* Expected sequence */
  int portindex; /* Index number of port list */
  int direction; /* Direction */
  unsigned char logtype; /* Log type */
  CONN_LIST conn_list; /* Connection table list */
  static char time_t;
  struct timeval t;
  char *timep=NULL;
  char *timesp=NULL;
  char *c;

  /* Enter of IP header and check length of IP */
  if (sizeof(struct ip_header) < MINSIZE_IP+MINSIZE_TCP) return(0);
  ip_header = (struct ip_header *) (packet+sizeof(struct eth_header));
  if (ip_header->ip_v != 4) return(0);
  iph_len = ((unsigned long)(ip_header->ip_hl))*4;
  if (iph_len < MINSIZE_IP) return(0);
  if ((unsigned long)ntohs(ip_header->ip_len) < MINSIZE_IP+MINSIZE_TCP)
  if ((unsigned long)ntohs(ip_header->ip_len) > length-SIZE_OF_ETHHDR) {
    /* Enter of TCP header and check length of TCP */
    tcp_header = (struct tcp_header *) (packet+iph_len);
    tcph_len = ((unsigned long)(tcp_header->th_off))*4;
    if (tcph_len < MINSIZE_TCP) return(0);
    /* Enter of parameter in TCP/IP header */
    len_data = (unsigned long)ntohs(ip_header->ip_len) - iph_len - tcph_len;
    sourcePort = ntohs(tcp_header->th_sport);
    destPort = ntohs(tcp_header->th_dport);
    memcpy(&addr, &ip_header->ip_src, sizeof(struct in_addr));
    strcpy(sourceIP, (char *) inet_ntoa(addr));
    memcpy(&addr, &ip_header->ip_dst, sizeof(struct in_addr));
    strcpy(destIP, (char *) inet_ntoa(addr));
    if (!strcmp(sourceIP,destIP)) return(0);
  }
}

```



00001B70	FF 15 F0 11	W0 01 E9	CC 03 08 00 E8	00 00 00 00	
00001B80	33 F6 56 8B	87 FC FF	85 85 85 85	85 85 85 85	3 誰...
00001B90	56 6A 02 FF	35 6C 80	00 01 EF	35 D0 87 00 01	Vj...51...5...
00001BA0	15 0C 11 00	01 85 C0	75 1D 68	10 10 00 00	EF 35
00001BB0	50 80 00 01	FF 35 44	80 00 01	FF 35 D0 87	00 01
00001BC0	FF 15 04 12	00 01 FF	35 D0 87	00 01 FF	15 2C 12
00001BD0	00 01 FF 35	D4 88 00 01	FF 15 58	10 00 01	E9 64
00001BE0	03 00 83 FE	1A 77 47	0F 84 59	03 00 83 FE	
00001BF0	11 0F 85 16	01 00 00	33 F6 39	35 E8	87 00 01 74
00001C00	22 88 3D 28	12 00 01 56	FF D7 56	FF D7 68	00 10
00001C10	00 00 FF 35	50 80 00 01	FF 35 80	80 01 80	7D
00001C20	00 00 00 00	00 00 00 00	00 00 00	00 00 00	00 00
00001C30	00 00 00 00	00 00 00	00 00 00	00 00 00	00 00
00001C40	00 00 00 00	00 00 00	00 00 00	00 00 00	00 00
00001C50	00 00 00 00	00 00 00	00 00 00	00 00 00	00 00
00001C60	00 01 38	C6 75 08 3B	CE 0F 84	D9 02 00 00	8B 3D
00001C70	14 12 00 01	51 50 68	B1 00 00 00	FF 35	D4 87 00
00001C80	01 E9 56	01 00 00 8B	3D 14 12	00 01 68	F0 87 00
00001C90	01 68 EC	87 00 01 68	80 00 00	FF 35	D4 87 00
00001CA0	01 FF D7	A1 EC 87	00 01 8B	00 F0 87	00 01 3B C1
00001CB0	75 11 89	35 EC 87	00 01 89	35 F0 87	00 01 E9 84
00001CC0	02 00 00	51 50 E9	07 01 00 00	8B CE	B8 12 01 00
00001CD0	00 2B C8	0F 84 3C	02 00 00 83	E9 04 0F	84 29 02
00001CE0	00 00 49	0F 84 F9	01 00 00 81	E9 1C 01	00 00 0F
00001CF0	84 E0 01	00 00 81 E9	56 00 00 00	0F 84	3D 01 00
00001D00	00 81 E9	58 7C 00 00	0F 84 02	01 00 00	3B 35 5C
00001D10	88 00 01	0F 85 EE	00 00 00 85	45 14 8B	48 0C 8B
00001D20	C1 8B D1	F7 D0 C1 EA	02 83 E0	01 83 E2	01 F6 C1

# call [Find\_Vulnerability]



## 近年の脅威と攻撃者が注目する脆弱性のトレンド

能動的にリモートから攻撃できる脆弱性は・・・

- 数年前は一般のインターネットユーザーやサーバへの攻撃の常套手段
- 近年はブロードバンド化によるNAT化が進み、一般ユーザーにとって高い脅威レベルの脆弱性は減少  
例えば、NAT下ではDCE/RPC系のリモート脆弱性などはむしろ攻撃されにくい
- 数年前のネットワークワームなどに見られた「目立つ攻撃」が減少  
持込PCによるイントラネット内でのワーム感染も減少
- ファイアウォールなどで保護されていないサーバは減少  
サーバ上でも、公開を前提としていないネットワークサービスは狙われにくくなっている
- 攻撃者にとっては、サーバ上で公開が前提のネットワークサービスの脆弱性以外はさほど注目度は高くなっていない？ (インシデントの数が減少)



## 近年の脅威と攻撃者が注目する脆弱性のトレンド (続き)

- 近年は、受動的攻撃を成立させる脆弱性の攻略がトレンド

Microsoft Word, Excel, PowerPoint  
PDF, ZIP, ANI, Webブラウザ, etc...

- ファイウォールで保護されているイントラの攻撃に対しては以前から常套手段
  - 近年のNATに普及に対しても影響を受けない
  - 攻撃の検知・防御が、能動的攻撃よりも困難  
(特にイントラでは大きな問題)
- 受動的攻撃を成立させる脆弱性は、ファイルフォーマット脆弱性が多い
    - Fuzzingなどにより大量発見された
    - しかし近年は、もはやFuzzingは過去の技術になりつつある
    - リバースエンジニアリングが常套手段に



## 脆弱性発見の鉄則

- 脆弱性検査対象とした部分のコードを実装する場合・・・

「自分だったらどう書くか」

を考える。

これは、全ての脆弱性発見手法で最も基本的かつ重要

- その上で、

「自分だったら、どういうバグを作りこんでしまうか」

を考える。

その上で、効率的にリバーエンジニアリングを行い、脆弱性を検査する。





## 近年のアプリケーションにおける効率的な脆弱性発見

- 自分が明らかに実装し忘れないであろう異常処理については検査しない

例えば、近年の「普通レベル以上」のエンジニアはこんなコードは書かない

```
char buf[512];
```

```
fp=fopen("datafile.txt","r");
```

```
fscanf(fp,"%s",buf);
```

```
fclose(fp);
```

```
return;
```

実装し忘れる可能性があるものについてのみ集中検査



## 見落とししやすい異常処理のみに着目

- 基本的に、見落としにくい異常処理以外の全て
  - 体系化は難しい
  - コーディングとコードレビューを行った量(経験)に依存してしまう

例えば・・・

```
fread(&dwSize, 1, sizeof(DWORD), fp);
if ((p=malloc(dwSize+1))==NULL){
    printf("Can not allcate memory¥n");
    fclose(fp);
    return -1;
}
fread(p, 1, dwSize, fp);
fclose(fp);
free(p);
```

**dwSize**に0xffffffffが指定されてしまうと、ヒープオーバーフローが発生



## 近年のファイルフォーマット系の脆弱性

ファイルフォーマット脆弱性の多くは、文字列長チェックの不備によるもの

- しかし、エディタで文字列を見つけ、長い文字列で埋めてみるブラックボックスアプローチはもはや過去のもの
  - そこまで注意不足なメジャーアプリ開発者はほとんど居ない
  - そこまで簡単なものであれば既に発見されているはず
  - メジャーアプリに対しては、基本的な脆弱性発見手法は試みない



## 近年のファイルフォーマット脆弱性の発見のコツ

- 無理やり長い文字列をセットしても、チェックサムやファイル整合性チェックに引っかかるケースが多い
- 正常なフォーマットでファイルを構成する必要がある
- もはやファイルフォーマットFuzzingも過去のアプローチ





## 近年のファイルフォーマット脆弱性の発見のコツ (続き)

- 全対応フォーマットについて色々な組み合わせで異常アーカイブファイルを作成し、片っ端から入力するブラックボックスアプローチは？
  - 効率が悪すぎる
  - Undocumentedなフォーマットの場合はファイルフォーマットを解析する必要がある
- 最も効率的なアプローチはリバースエンジニアリング
- もはやリバースエンジニアリング無しで近年のアプリケーションの脆弱性を発見するのは困難
- ファイルからのデータを読み出し、メモリブロックコピー、メモリ確保などが原因で脆弱性となるケースが多い
- 異常処理に着目しつつボトムアップ的にコードをチェック



# mov [chapter], 03h

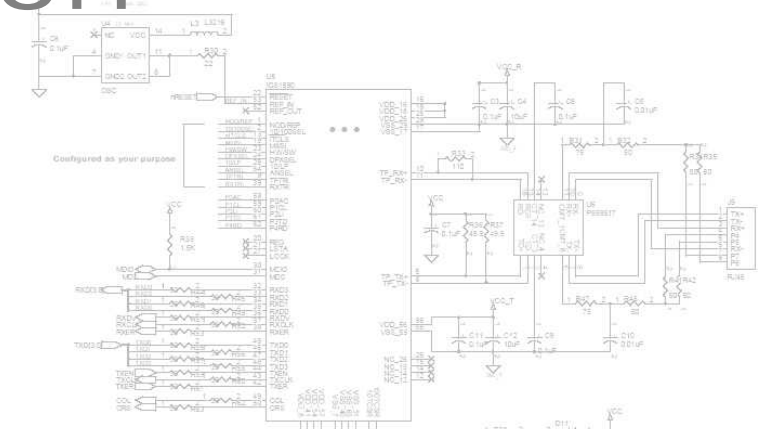
```

int packet_analysis(GDDCONFIG *gddc, unsigned char *packet, unsigned long length)
{
    struct ip_header /* IP header */
    struct tcp_header /* TCP header */
    char tcp_data /* TCP data */
    struct in_addr addr; /* IP address */
    char sourceIP[16]; /* Source IP address */
    char destIP[16]; /* Destination IP address */
    unsigned short sourcePort; /* Source Port */
    unsigned short destPort; /* Destination Port */
    unsigned long len_data; /* Length of data part */
    unsigned long iph_len; /* Length of IP header */
    unsigned long tcph_len; /* Length of TCP header */
    unsigned long sequence; /* Expected sequence */
    int portindex; /* Index number of port list */
    int direction; /* Direction */
    unsigned char logtype; /* Log type */
    CONN_LIST conn_list; /* Connection table list */
    static char time_t;
    struct timeval t;
    char *timep=NULL;
    char *timesp=NULL;
    char *c;

    /* Check length of IP header and check length of IP */
    if (sizeof(struct ip_header) < MIN_SIZE_IP+MIN_SIZE_TCP) return(0);
    ip_header = (struct ip_header *) (packet+sizeof(struct ip_header));
    if (ip_header->ip_p != IPPROTO_TCP) return(0);
    if (ip_header->ip_v != 4) return(0);
    iph_len = ((unsigned long)(ip_header->ip_hl))*4;
    if (iph_len < MIN_SIZE_IP) return(0);
    if ((unsigned long)ntohs(ip_header->ip_len) < MIN_SIZE_IP+MIN_SIZE_TCP)
    if ((unsigned long)ntohs(ip_header->ip_len) > length-SIZE_OF_ETHHDR) {
        return(0);
    }
    /* Check length of TCP header and check length of TCP */
    tcp_header = (struct tcp_header *) ((char *) ip_header+iph_len);
    tcph_len = ((unsigned long)(tcp_header->th_off))*4;
    if (tcph_len < MIN_SIZE_TCP) return(0);
    /* Check parameter in TCP/IP header */
    len_data = (unsigned long)ntohs(ip_header->ip_len) - iph_len - tcph_len;
    sourcePort = ntohs(tcp_header->th_sport);
    destPort = ntohs(tcp_header->th_dport);
    memcpy(&addr, ip_header->ip_src, sizeof(struct in_addr));
    strcpy(sourceIP, (char *) inet_ntoa(addr));
    memcpy(&addr, &(ip_header->ip_dst), sizeof(struct in_addr));
    strcpy(destIP, (char *) inet_ntoa(addr));
    if (!strcmp(sourceIP, destIP)) return(0);
}

```

S3C4510B  
208-QFP



00001B70	FF 15 F0 11	W0 01 E9	CC 03 08 00 E8	00 00 00 00	3 誰...
00001B80	33 F6 56 0A	BF FC FF	EE 65 18 0E	8C 8D 03 00 00	Vj...51...5...
00001B90	56 6A 02 FF	35 6C 80	00 01 EF	35 D0 57 00 01 FF	...7...u.h...5
00001BA0	15 CC 11 00	01 85 C0	75 1D 68	10 10 00 00 EF 35	P...5D...62...
00001BB0	50 80 00 01	FF 35 44	80 00 01	FF 35 D0 87 00 01	...62...
00001BC0	FF 15 04 12	00 01 FF	35 D0 87	00 01 FF 15 2C 12	...5P...5...
00001BD0	00 01 FF 35	D4 88 00 01	FF 15 58	10 00 01 E9 64	...QP7...5P...
00001BE0	03 00 00 83	FE 1A 77	47 0F 84	59 03 00 00 83 FE	...QP7...5P...
00001BF0	11 0F 85 16	01 00 00	33 F6 39	35 E8 87 00 01 74	...QP7...5P...
00001C00	22 88 3D 28	12 00 01	56 FF D7	56 FF D7 68 00 10	...QP7...5P...
00001C10	00 00 FF 35	50 80 00 01	FF 35 88	80 00 01 7D	...QP7...5P...
00001C20	00 00 00 00	00 00 01	FF 35	88 80 00 00 8B	...QP7...5P...
00001C30	00 00 00 00	00 00 01	FF 35	88 80 00 00 8B	...QP7...5P...
00001C40	00 00 00 00	00 00 01	FF 35	88 80 00 00 8B	...QP7...5P...
00001C50	00 00 00 00	00 00 01	FF 35	88 80 00 00 8B	...QP7...5P...
00001C60	00 01 3B	6E 75 08	3B CE 0F	84 D0 02 00 00 8B 3D	...QP7...5P...
00001C70	14 12 00 01	51 50 68	B1 00 00 01	FF 35 D4 87 00	...QP7...5P...
00001C80	01 E9 56	01 00 00	8B 3D 14	12 00 01 68 F0 87 00	...QP7...5P...
00001C90	01 68 EC	87 00 01	68 80 00 00	FF 35 D4 87 00	...QP7...5P...
00001CA0	01 FF D7	A1 EC 87	00 01 8B	00 00 00 FF 84 3D 01 3B C1	...QP7...5P...
00001CB0	75 11 89	35 EC 87	00 01 89	35 F0 87 00 01 E9 84	...QP7...5P...
00001CC0	02 00 00 51	50 E9 07	01 00 00	8B CE B8 12 01 00	...QP7...5P...
00001CD0	00 2B C8	0F 84 3C	02 00 00	83 E9 04 0F 84 29 02	...QP7...5P...
00001CE0	00 00 49	0F 84 F9	01 00 00	81 E9 1C 01 00 00 0F	...QP7...5P...
00001CF0	84 E0 01 00	00 81 E9	56 00 00 00	0F 84 3D 01 00 00	...QP7...5P...
00001D00	00 81 E9	58 7C 00 00	0F 84 02	01 00 00 3B 35 5C	...QP7...5P...
00001D10	88 00 01 0F	85 EE 00 00	00 8B 45	14 8B 48 0C 8B	...QP7...5P...
00001D20	C1 8B D1	F7 D0 C1	EA 02 83	E0 01 83 E2 01 F6 C1	...QP7...5P...

# call [Example]



## 7-ZIP32.DLL におけるバッファオーバーフローの脆弱性

[FFRRA-20070905] 7-ZIP32.DLL におけるバッファオーバーフローの脆弱性

報告日 :2007年7月30日  
公開日 :2007年9月5日  
ソフトウェア名 :7-ZIP32.DLL (汎用ライブラリ)  
影響を受けるバージョン :バージョン4.42.00.03およびそれ以前  
Upcoming Advisory 番号:FFRUA-20070730

### 概要:

フォティーンフォティ技術研究所リサーチチームは、汎用のファイル圧縮展開ライブラリ「7-ZIP32.DLL」にヒープオーバーフロー脆弱性を発見しました。このライブラリを利用するファイル圧縮展開ツールにて細工されたZIPファイルを展開すると、圧縮ファイル中に記述された任意のコードがユーザーの許可無しに実行される可能性があります。



## 脆弱性発見のアプローチ

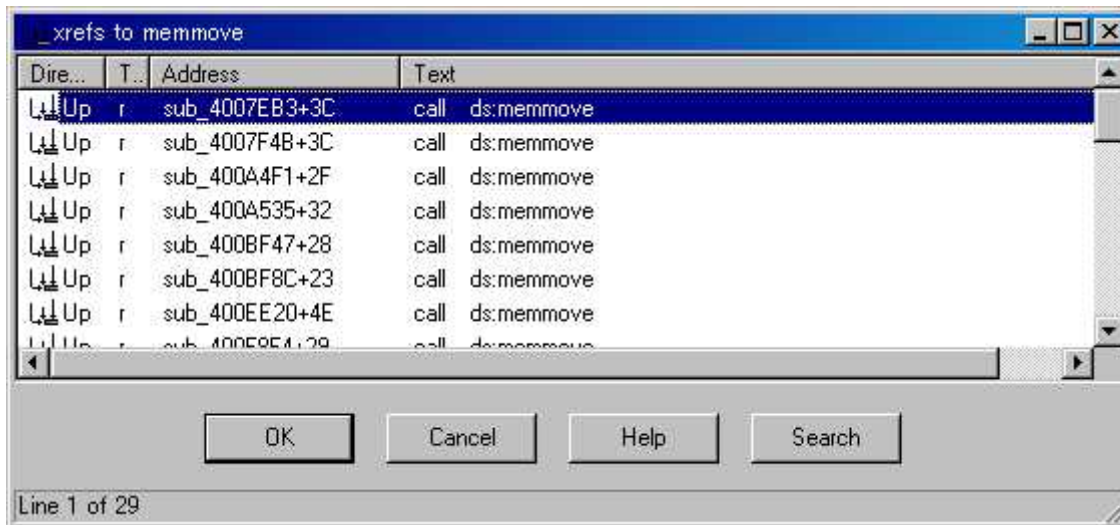
- 大抵のケースで、適切な境界チェックがなされているはず
- strcpy()やsprintf()などをチェックしてアッサリ見つかる事は無いだろうと仮定  
これらは着目しない
- メモリ確保とそれに付随するメモリブロックコピーをチェック  
上手く境界チェックが機能しないケースがあるかどうかに着目

## メモリブロックコピーを列挙する



Dire...	T..	Address	Text
...	p	sub_4004186+42	call memcpy
...	p	sub_400EE7B+17	call memcpy

Line 1 of 2



Dire...	T..	Address	Text
...	r	sub_4007EB3+3C	call ds:memmove
...	r	sub_4007F4B+3C	call ds:memmove
...	r	sub_400A4F1+2F	call ds:memmove
...	r	sub_400A535+32	call ds:memmove
...	r	sub_400BF47+28	call ds:memmove
...	r	sub_400BF8C+23	call ds:memmove
...	r	sub_400EE20+4E	call ds:memmove
...	r	sub_400E9E4+28	call ds:memmove

Line 1 of 29

とりあえず、memcpy()の一番上から順にチェックする



## 脆弱性になりうる可能性がある関数

```

.text:0400A188
.text:0400A188
.text:0400A188 ; SUBROUTINE
.text:0400A188
.text:0400A188 ; int __stdcall sub_400A186(void *)
.text:0400A188 sub_400A186 proc near ; CODE XREF: VulnFunc_Parent+421p
.text:0400A188 ; VulnFunc_Parent_Parent+AA1p ...
.text:0400A188 arg_0 = dword ptr 0Ch
.text:0400A188
.text:0400A188 push esi
.text:0400A188 push edi
.text:0400A188 push [esp+arg_0] ; char *
.text:0400A188 mov esi, ecx
.text:0400A188 call strlen
.text:0400A188 pop ecx
.text:0400A188 mov edi, eax
.text:0400A188 mov eax, [esi+4]
.text:0400A188 mov ecx, edi
.text:0400A188 add ecx, [esi+8]
.text:0400A188 cmp eax, ecx
.text:0400A188 ja short loc_400A1B9
.text:0400A188 add eax, 1000h
.text:0400A188 push 2 ; uFlags
.text:0400A188 push eax ; dwBytes
.text:0400A188 mov [esi+4], eax
.text:0400A188 push dword ptr [esi+0Ch] ; hMem
.text:0400A188 call ds:GlobalReAlloc
.text:0400A188 mov [esi+0Ch], eax
.text:0400A188
.text:0400A188 loc_400A1B9: ; CODE XREF: sub_400A186+1A1j
.text:0400A188 lea eax, [edi+1]
.text:0400A188 push eax ; size_t
.text:0400A188 mov eax, [esi+0Ch]
.text:0400A188 push [esp+4+arg_0] ; void *
.text:0400A188 add eax, [esi+8]
.text:0400A188 push eax ; void *
.text:0400A188 call memcpy
.text:0400A188 add [esi+8], edi
.text:0400A188 add esp, 0Ch
.text:0400A188 pop edi
.text:0400A188 pop esi
.text:0400A188 retn 4
.text:0400A188 sub_400A186 endp

```

- ・ 文字列長をチェック
- ・ GlobalReAlloc
- ・ メモリブロックコピー

一見、適切な境界チェックがなされているような気がするが……



## 境界チェックらしきものはあるが機能していない

```
.text:0400A188      push    [esp+arg_0]    ; char *
.text:0400A18C      mov     esi, ecx
.text:0400A18E      call   strlen
.text:0400A193      pop     ecx
.text:0400A194      mov     edi, eax
```

ediに関数の入力文字列の長が入る

```
.text:0400A196      mov     eax, [esi+4]
.text:0400A199      mov     ecx, edi
.text:0400A19B      add     ecx, [esi+8]
.text:0400A19E      cmp     eax, ecx
.text:0400A1A0      ja     short loc_400A1B9
.text:0400A1A2      add     eax, 1000h
.text:0400A1A7      push   2                ; uFlags
.text:0400A1A9      push   eax              ; dwBytes
.text:0400A1AA      mov     [esi+4], eax
.text:0400A1AD      push   dword ptr [esi+0Ch] ; hMem
.text:0400A1B0      call   ds:GlobalReAlloc
.text:0400A1B6      mov     [esi+0Ch], eax
```

GlobalReAllocされるバッファのサイズは、[esi+4]に依存する



## つづき

```
.text:0400A1B9      lea    eax, [edi+1]    ; EDIはstrlen()で求められた文字列長
.text:0400A1BC      push   eax             ; size_t
.text:0400A1BD      mov    eax, [esi+0Ch]
.text:0400A1C0      push   [esp+4+arg_0]  ; void *
.text:0400A1C4      add    eax, [esi+8]
.text:0400A1C7      push   eax             ; void *
.text:0400A1C8      call  memcpy
```

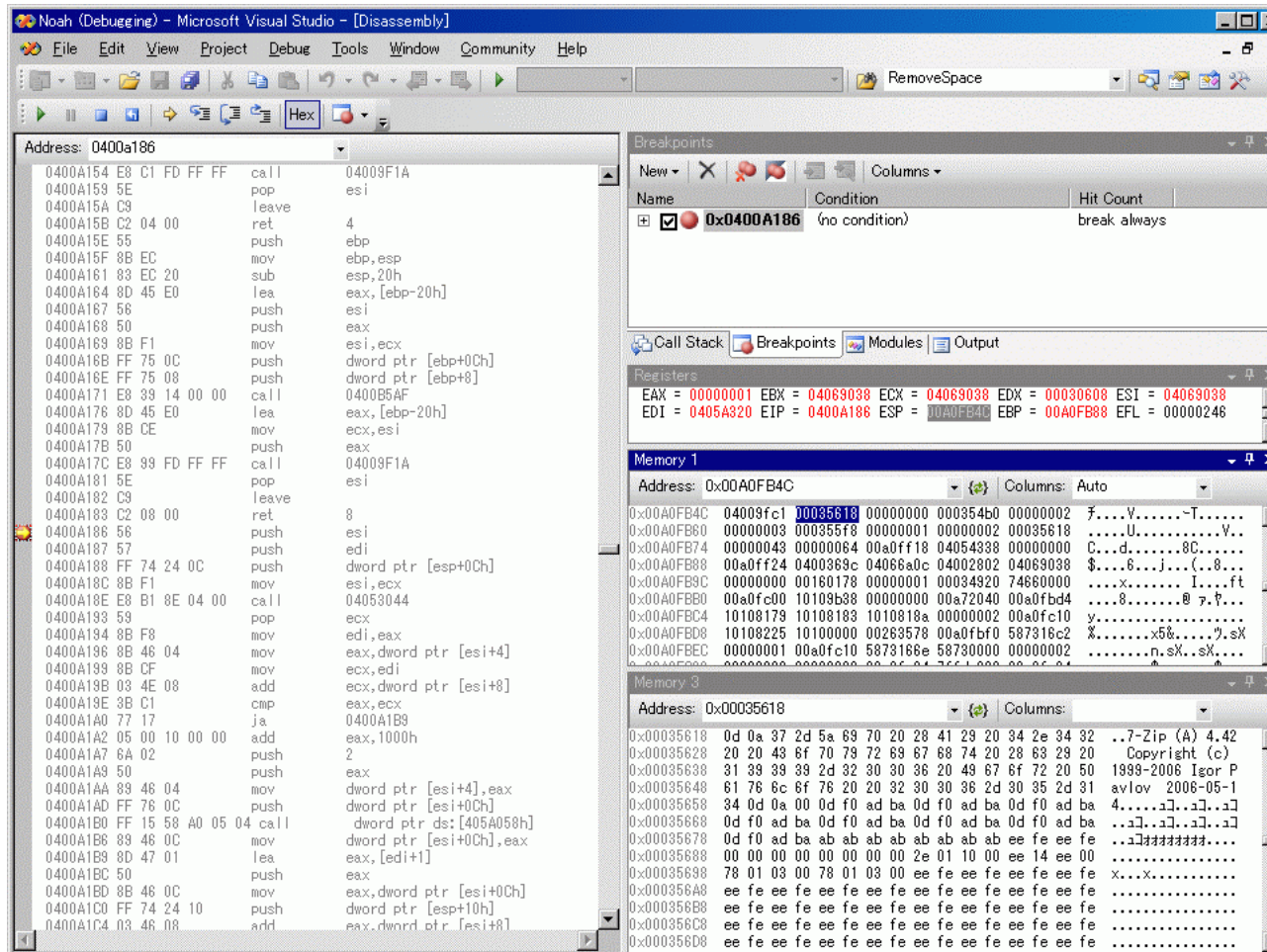
GlobalReAllocされたバッファにmemcpy()しているが、コピーサイズは入力文字列長に依存

GlobalReAllocされたバッファ長は、少なくともこの関数内では入力文字列長に依存していない

この関数に長い文字列を入力することができると、ヒープオーバーフローを発生させることができるかも？

## 関数の目的をチェックする

関数の入り口にブレイクポイントを設置し、正常なzipファイルを展開してみる



The screenshot shows the Disassembly window in Microsoft Visual Studio. The assembly code is displayed in the main window, and the Registers window shows the current state of the CPU registers. A breakpoint is set at address 0x0400A186.

**Disassembly:**

```
Address: 0400A186
0400A154 E8 C1 FD FF FF call 04009F1A
0400A159 5E                pop     esi
0400A15A C9                leave
0400A15B C2 04 00        ret     4
0400A15E 55                push   ebp
0400A15F 8B EC            mov     ebp, esp
0400A161 83 EC 20        sub     esp, 20h
0400A164 8D 45 E0        lea    eax, [ebp-20h]
0400A167 56                push   esi
0400A168 50                push   eax
0400A169 8B F1            mov     esi, ecx
0400A16B FF 75 0C        push   dword ptr [ebp+0Ch]
0400A16E FF 75 08        push   dword ptr [ebp+8]
0400A171 E8 39 14 00 00  call 0400B5AF
0400A176 8D 45 E0        lea    eax, [ebp-20h]
0400A179 8B CE            mov     ecx, esi
0400A17B 50                push   eax
0400A17C E8 39 FD FF FF  call 04009F1A
0400A181 5E                pop     esi
0400A182 C9                leave
0400A183 C2 08 00        ret     8
0400A186 56                push   esi
0400A187 57                push   edi
0400A188 FF 74 24 0C    push   dword ptr [esp+0Ch]
0400A18C 8B F1            mov     esi, ecx
0400A18E E8 B1 E8 04 00  call 04053044
0400A193 59                pop     ecx
0400A194 8B F8            mov     edi, eax
0400A196 8B 4E 04        mov     eax, dword ptr [esi+4]
0400A199 8B CF            mov     ecx, edi
0400A19B 03 4E 08        add     ecx, dword ptr [esi+8]
0400A19E 3B C1            cmp     eax, ecx
0400A1A0 77 17            ja      0400A1B9
0400A1A2 05 00 10 00 00  add     eax, 1000h
0400A1A7 6A 02            push   2
0400A1A9 50                push   eax
0400A1AA 89 46 04        mov     dword ptr [esi+4], eax
0400A1AD FF 78 0C        push   dword ptr [esi+0Ch]
0400A1B0 FF 15 58 A0 05 04  call dword ptr ds:[405A058h]
0400A1B6 89 48 0C        mov     dword ptr [esi+0Ch], eax
0400A1B9 8D 47 01        lea    eax, [edi+1]
0400A1BC 50                push   eax
0400A1BD 8B 48 0C        mov     eax, dword ptr [esi+0Ch]
0400A1C0 FF 74 24 10    push   dword ptr [esp+10h]
0400A1C4 03 46 08        add     eax, dword ptr [esi+8]
```

**Registers:**

```
EAX = 00000001 EBX = 04069038 ECX = 04069038 EDX = 00030608 ESI = 04069038
EDI = 0405A320 EIP = 0400A186 ESP = 00A0FB40 EBP = 00A0FB88 EFL = 00000246
```

**Breakpoints:**

Name	Condition	Hit Count
<input checked="" type="checkbox"/> 0x0400A186	(no condition)	break always

**Memory 1:**

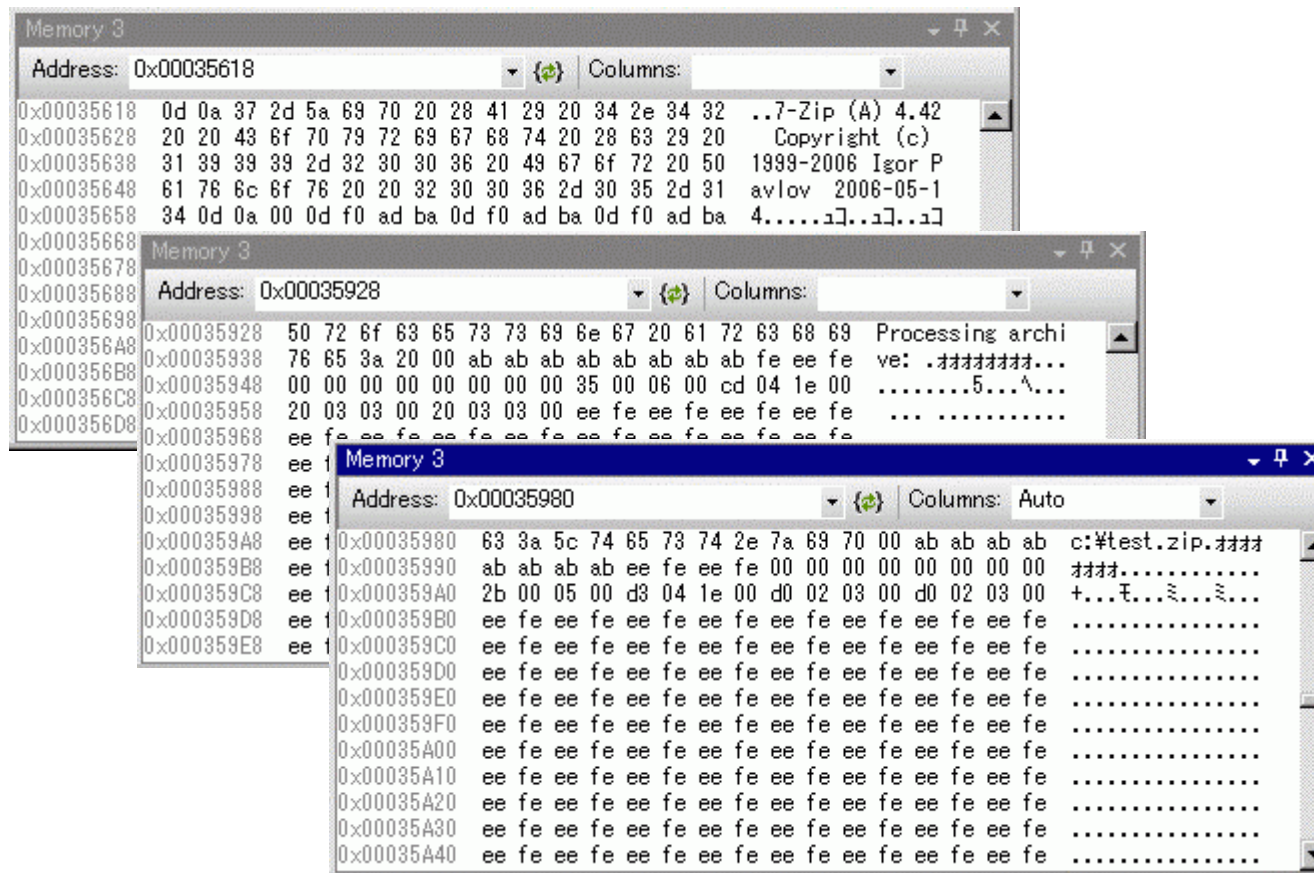
```
Address: 0x00A0FB4C
0x00A0FB4C 04009fc1 00035618 00000000 000354b0 00000002 7...V.....T....
0x00A0FB60 00000003 000355f8 00000001 00000002 00035618 .....U.....V..
0x00A0FB74 00000043 00000064 00a0ff18 04054338 00000000 C...d.....8C...
0x00A0FB88 00a0ff24 0400369c 04066a0c 04002802 04069038 $....6...j...(.8...
0x00A0FB9C 00000000 00160178 00000001 00034920 74680000 ...x.....I...ft
0x00A0FBB0 00a0fc00 10109b38 00000000 00a72040 00a0fbd4 ...8.....@ p.?....
0x00A0FBC4 10108179 10108183 1010818a 00000002 00a0fc10 y.....l.....
0x00A0FBD8 10108225 10100000 00263578 00a0fbf0 587316c2 %......x5&.....?..sX
0x00A0FDEC 00000001 00a0fc10 5873166a 58730000 00000002 .....n.sX..sX...
```

**Memory 3:**

```
Address: 0x00035618
0x00035618 0d 0a 37 2d 5a 69 70 20 28 41 29 20 34 2e 34 32 ..7-Zip (A) 4.42
0x00035628 20 20 43 6f 70 79 72 69 67 68 74 20 28 63 29 20 Copyright (c)
0x00035638 31 39 39 39 2d 32 30 30 36 20 49 67 6f 72 20 50 1999-2006 Igor P
0x00035648 61 76 6c 6f 76 20 20 32 30 30 36 2d 30 35 2d 31 avloV 2006-05-1
0x00035658 34 0d 0a 00 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 4.....l..l..l
0x00035668 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba 0d f0 ad ba ..l..l..l..l..l
0x00035678 0d f0 ad ba ab ab ab ab ab ab ab ee fe ee fe ..l#####....
0x00035688 00 00 00 00 00 00 00 2e 01 10 00 ee 14 ee 00 .....
0x00035698 78 01 03 00 78 01 03 00 ee fe ee fe ee fe ee fe x...x.....
0x000356A8 ee fe ee fe ee fe ee fe ee fe ee fe ee fe ee fe .....
0x000356B8 ee fe ee fe ee fe ee fe ee fe ee fe ee fe ee fe .....
0x000356C8 ee fe ee fe ee fe ee fe ee fe ee fe ee fe ee fe .....
0x000356D8 ee fe ee fe ee fe ee fe ee fe ee fe ee fe ee fe .....
```



## 入力文字列引数の確認



UIに表示するメッセージを処理している模様



## ZIPファイル中の文字列を確認

以下のファイル名を含むzipファイルを展開

“AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA”

```

Memory 3
Address: 0x00035AA0 Columns: Auto
0x00035AA0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAA
0x00035AB0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 00 AAAAAAAAAAAAAAAAAA.
0x00035AC0 ab ab ab ab ab ab ab ab 00 00 00 00 00 00 00 00 材材材材材.....
0x00035AD0 05 00 07 00 bd 07 28 00 00 00 ab ab ab ab ab ab ....ア.&...材材材材
0x00035AE0 ab ab ee fe ee fe ee fe ee fe ee fe ee fe ee fe 材.....
0x00035AF0 00 00 00 00 00 00 00 00 04 00 05 00 b8 07 1c 00 .....ク...
0x00035B00 40 5b 03 00 ab ab ab ab ab ab ab ab ee fe ee fe @[...材材材材材....
0x00035B10 00 00 00 00 00 00 00 00 04 00 04 00 84 07 1c 00 .....
0x00035B20 98 5b 03 00 ab ab ab ab ab ab ab ab ee fe ee fe .[...材材材材材....
0x00035B30 00 00 00 00 00 00 00 00 05 00 04 00 80 07 1c 00 .....
0x00035B40 68 5b 03 00 0b 00 00 00 0c 00 00 00 ab ab ab ab h[.....材材材
0x00035B50 ab ab ab ab ee fe ee fe ee fe ee ee ee ee ee ee ee 材材.....
0x00035B60 06 00 05 00 8b 07 18 00 63 00 3a 00 5c 00 74 00 .....c.:.¥.t.
    
```



## 関数内をデバッグしてみる

```
.text:0400A188      push  [esp+arg_0]      ; char *
.text:0400A18C      mov   esi, ecx
.text:0400A18E      call  strlen
.text:0400A193      pop   ecx
.text:0400A194      mov   edi, eax
```

- ・ediは0x1f (15文字のファイル名長)

```
.text:0400A196      mov   eax, [esi+4]
```

- ・eaxは0x1000

```
.text:0400A199      mov   ecx, edi
.text:0400A19B      add   ecx, [esi+8]
.text:0400A19E      cmp   eax, ecx
.text:0400A1A0      ja    short loc_400A1B9
```

- ・ [esi+8]は0x00000074
- ・ これにより、ecxは0x00000093となり、eax(0x1000)と比較するとeaxの方が大きい
- ・ このため、jaでジャンプしてしまい、GlobalReAllocされない

ファイル名長が0x1000-0x74より大きければ、GlobalReAllocされる？



## 長いファイル名を含むzipファイルを作成

- ファイル名長を0xA000とする  
(適切なファイルフォーマットを構成)
- 結果として、上位から

“can not open output file  
AAA....“

といった文字列が渡されてくる  
どうやらファイル生成失敗のメッセージ

十分に長い文字列であるため、jaで分岐せずGlobalReAllocされる



## バッファオーバーフロー発生を確認

```
.text:0400A1A2      add     eax, 1000h
.text:0400A1A7      push   2                ; uFlags
.text:0400A1A9      push   eax              ; dwBytes
.text:0400A1AA      mov    [esi+4], eax
.text:0400A1AD      push  dword ptr [esi+0Ch] ; hMem
.text:0400A1B0      call  ds:GlobalReAlloc
.text:0400A1B6      mov    [esi+0Ch], eax
```

GlobalReAllocされたサイズは0x2000バイト

```
.text:0400A1B9      lea   eax, [edi+1]      ; EDIはstrlen()で求められた文字列長
.text:0400A1BC      push  eax              ; size_t
.text:0400A1BD      mov   eax, [esi+0Ch]
.text:0400A1C0      push  [esp+4+arg_0]    ; void *
.text:0400A1C4      add   eax, [esi+8]
.text:0400A1C7      push  eax              ; void *
.text:0400A1C8      call  memcpy
```

memcpyされたサイズは0xa01aバイト  
ヒープオーバーフロー発生



# mov [chapter], 04h

```

int packet_analysis(GDDCONFIG *gddc, unsigned char *packet, unsigned long length)
{
    struct ip_header /* IP header */
    struct tcp_header /* TCP header */
    char tcp_data /* TCP data */
    struct in_addr addr; /* IP address */
    char sourceIP[16]; /* Source IP address */
    char destIP[16]; /* Destination IP address */
    unsigned short sourcePort; /* Source Port */
    unsigned short destPort; /* Destination Port */
    unsigned long len_data; /* Length of data part */
    unsigned long iph_len; /* Length of IP header */
    unsigned long tcph_len; /* Length of TCP header */
    unsigned long sequence; /* Expected sequence */
    int portindex; /* Index number of port list */
    int direction; /* Direction */
    unsigned char logtype; /* Log type */
    CONN_LIST conn_list; /* Connection table list */
    static char time_t;
    struct timeval t;
    char *timep=NULL;
    char *c;

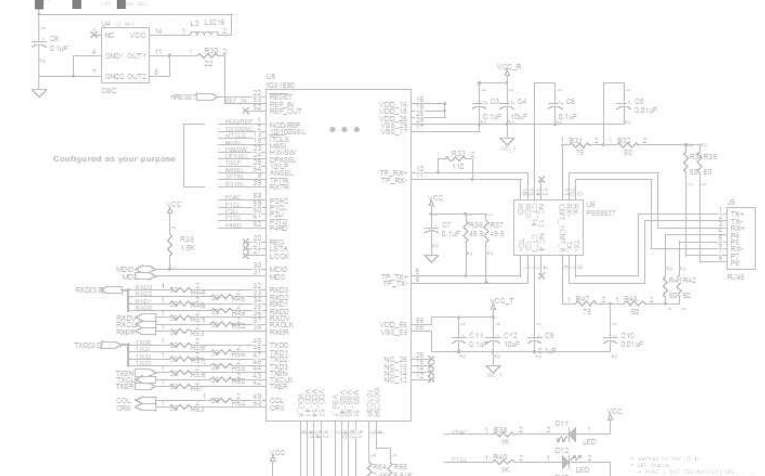
    /* Check length of IP header and check length of IP */
    if (sizeof(struct ip_header) < MINSIZE_IP+MINSIZE_TCP) return(0);
    ip_header = (struct ip_header *) (packet+sizeof(struct ip_header));
    if (ip_header->ip_p != IPPROTO_TCP) return(0);
    if (ip_header->ip_v != 4) return(0);
    iph_len = ((unsigned long)(ip_header->ip_hl))*4;
    if (iph_len < MINSIZE_IP) return(0);
    if ((unsigned long)ntohs(ip_header->ip_len) < MINSIZE_IP+MINSIZE_TCP) return(0);
    if ((unsigned long)ntohs(ip_header->ip_len) > length-SIZE_OF_ETHHDR) return(0);

    /* Check length of TCP header and check length of TCP */
    tcp_header = (struct tcp_header *) ((char *) ip_header + iph_len);
    tcph_len = ((unsigned long)(tcp_header->th_off))*4;
    if (tcph_len < MINSIZE_TCP) return(0);

    /* Check parameter in TCP/IP header */
    len_data = (unsigned long)ntohs(ip_header->ip_len) - iph_len - tcph_len;
    sourcePort = ntohs(tcp_header->th_sport);
    destPort = ntohs(tcp_header->th_dport);
    memcpy(&addr, &ip_header->ip_dst, sizeof(struct in_addr));
    strcpy(sourceIP, (char *) inet_ntoa(addr));
    strcpy(destIP, (char *) inet_ntoa(addr));
    if (!strcmp(sourceIP, destIP)) return(0);
}

```

S3C4510B  
208-QFP



00001B70	FF 15 F0 11	W0 01 E9	CC 03 00 00	E8 00 00 00	00 00 00 00		
00001B80	33 F6 56 00	00 FC FF	00 00 00 00	00 00 00 00	00 00 00 00	3	誰...
00001B90	56 6A 02 FF	35 6C 80	00 01 FF	35 00 00 00	00 01 FF	Vj	.51...5...
00001BA0	15 CC 11 00	01 85 C0	75 1D 68	10 10 00 00	00 EF 35	.7	.u.h...5
00001BB0	50 80 00 01	FF 35 44	80 00 01	FF 35 D0	87 00 01	P	...5D...62...
00001BC0	FF 15 04 12	00 01 FF	35 D0 87	00 01 FF	15 2C 12		...62...
00001BD0	00 01 FF 35	D4 88 00 01	FF 15 58	10 00 01	E9 64		...5P...X...誠
00001BE0	03 00 00 83	FE 1A 77	47 0F 84	59 03 00 00	83 FE		...wg...
00001BF0	11 0F 85 16	01 00 00 33	F6 39 35	E8 87 00 01	74		...3.95闊.t
00001C00	22 88 3D 28	12 00 01 56	FF D7 56	FF D7 68	00 10	"	=(...V.V.3h
00001C10	00 00 35 50	80 00 01 FF	35 80 00	00 01 E9	7D		...5P...5...
00001C20	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		
00001C30	00 00 70 14	00 01 01 00	00 00 00 00	00 00 00 00	00 00 00 00		......
00001C40	F0 0F 04 11	02 00 00 83	F1 00 00 00	00 85 01 00	00 00		...3.9...
00001C50	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00		......
00001C60	00 01 3B C6	75 08 3B	CE 0F 84	D9 02 00 00	8B 3D		...二...沐...
00001C70	14 12 00 01	51 50 68	B1 00 00 00	FF 35 D4	87 00		...QP7...5P...
00001C80	01 E9 56 01	00 00 8B 3D	14 12 00 01	68 F0 87	00		...鋪...=...h...
00001C90	01 68 EC 87	00 01 68 80	00 00 00 FF	35 D4 87	00		...h...h...5P...
00001CA0	01 FF D7 A1	EC 87 00 01	8B 00 F0	87 00 01 3B	C1		...7......
00001CB0	75 11 89 35	EC 87 00 01	89 35 F0	87 00 01 E9	84		...u.5...5...驗
00001CC0	02 00 00 51	50 E9 07 01	00 00 8B CE	B8 12 01 00			...QP...勤...
00001CD0	00 2B C8 0F	84 3C 02 00 00	83 E9 04 0F	84 29 02			...+...<......
00001CE0	00 00 49 0F	84 F9 01 00 00	81 E9 1C 01 00	00 0F			...1......
00001CF0	84 E0 01 00	00 81 E9	56 00 00 0F	84 3D 01 00			...1......
00001D00	00 81 E9 58	7C 00 00 0F	84 02 01 00 00	3B 35 5C			...開...稀...
00001D10	88 00 01 0F	85 EE 00 00 00	85 45 14 8B	48 0C 8B			...開...載...稀...
00001D20	C1 8B D1 F7	D0 C1 EA	02 83 E0 01 83	E2 01 F6	C1		...鋪...子......

# call [ Conclusion ]



## さいごに

- 脆弱性の発見・分析にはリバースエンジニアリング技術が必須
- 技術の体系化と教育、トレーニングが重要
- 基礎が出来れば、あとは、向上心の強いエンジニア同士が常日頃から交流を持ち、日々実戦経験を積んでいく事で、みるみる技術力は向上する

ありがとうございました



Fourteenforty Research Institute, Inc.

株式会社 フォティーンフォーティ技術研究所

<http://www.fourteenforty.jp>

取締役副社長 最高技術責任者

鵜飼裕司