RBS-2019-004

PS_NTS ActiveX Control ps_ntscrypto.dll Relative Distinguished Names Handling Buffer Overflows

Vulnerable Program Details

Details for tested products and versions:

Vendor:	SG Co Ltd.
Product:	PS_NTS ActiveX Control (PS_NTS.dll)
Version:	4.1.1.4
Component:	ps_ntscrypto.dll
Component Version:	4.1.1.3

NOTE: Other versions than the one listed above are likely affected.

Credits

Carsten Eiram, Risk Based Security
Twitter: @RiskBased

Impact

The PS_NTS ActiveX control bundles the ps_ntscrypto.dll library, which contains various buffer overflows that may allow an attacker to compromise a user's system.

Vulnerability Details

The ps_ntscrypto.dll library bundled with the PS_NTS ActiveX control exports various crypto functionality, where some of the functions of particular interest to these vulnerabilities are e.g. ps_req_auth_string() and ps_get_session_key(). The exported functions can be reached via methods provided by the ActiveX control such as PsWebClientReqAuth() and PsWebGetSKey().

The PsWebClientReqAuth() method for authenticating a client is defined as follows:

```
[id(0x0000000e), helpstring("method PsWebClientReqAuth")]
BSTR PsWebClientReqAuth(
                [in] BSTR inCname,
                [in] BSTR inPasswd,
                [in] BSTR inRdname);
```

The PsWebGetSKey() method for getting a session key is defined as follows:

```
[id(0x00000028), helpstring("method PsWebGetSKey")]
BSTR PsWebGetSKey([in] BSTR RDNAME);
```

In both cases, the argument of interest is ‘inRdname’ or ‘RDNAME’, which is intended to supply a list of relative distinguished names.

When either of these two methods is called, the function in PS_NTS.dll that is responsible for handling the call eventually transfers control to the corresponding exported function in ps_ntscrypto.dll along with the supplied arguments. Both functions end up calling a function to parse the supplied relative distinguished name string. This function allocates a 201 byte buffer on the heap and checks if the supplied string starts with “cn=”.

```
.text:022D5FA0 ; int __cdecl ConstructRDName(char *pszinRdname)
.text:022D5FA0 ConstructRDName proc near               ; CODE XREF: sub_22D1CD0+53p
.text:022D5FA0                                         ; sub_22D1D40+53p ...
.text:022D5FA0
.text:022D5FA0 iCount          = dword ptr -978h
.text:022D5FA0 szC             = byte ptr -974h        ; char[4]
.text:022D5FA0 iUnk1           = dword ptr -970h
.text:022D5FA0 wUnk2           = word ptr -96Ch
.text:022D5FA0 lpszDest        = dword ptr -968h
.text:022D5FA0 szOU            = byte ptr -964h        ; char[48]
.text:022D5FA0 anonymous_0     = word ptr -934h
.text:022D5FA0 szO             = byte ptr -930h        ; char[52]
.text:022D5FA0 szCN            = byte ptr -8FCh        ; char[100]
.text:022D5FA0 var_898         = byte ptr -898h        ; char[200]
.text:022D5FA0 var_7D0         = byte ptr -7D0h        ; char[2000]
.text:022D5FA0 pszinRdname     = dword ptr  4
.text:022D5FA0
.text:022D5FA0                 sub     esp, 978h
.text:022D5FA6                 push    ebx
.text:022D5FA7                 push    ebp
.text:022D5FA8                 push    esi
.text:022D5FA9                 push    edi
.text:022D5FAA                 mov     ecx, 19h
.text:022D5FAF                 xor     eax, eax
.text:022D5FB1                 lea     edi, [esp+988h+szCN] ; char[100]
.text:022D5FB8                 mov     ebx, [esp+988h+pszinRdname]
.text:022D5FBF                 rep stosd
.text:022D5FC1                 mov     dword ptr [esp+988h+szC], eax ; char[4]
.text:022D5FC5                 mov     ecx, 0Ch
.text:022D5FCA                 mov     [esp+988h+iUnk1], eax
.text:022D5FCE                 lea     edi, [esp+988h+szO] ; char[52]
.text:022D5FD2                 mov     [esp+988h+wUnk2], ax
.text:022D5FD7                 rep stosd
.text:022D5FD9                 stosw
.text:022D5FDB                 mov     ecx, 0Ch
.text:022D5FE0                 xor     eax, eax
.text:022D5FE2                 lea     edi, [esp+988h+szOU] ; char[48]
.text:022D5FE6                 rep stosd
.text:022D5FE8                 stosw
```

```
.text:022D5FEA                  mov     ecx, 32h
.text:022D5FEF                  xor     eax, eax
.text:022D5FF1                  lea     edi, [esp+988h+var_898] ; char[200]
.text:022D5FF8                  test    ebx, ebx
.text:022D5FFA                  rep stosd
.text:022D5FFC                  jnz     short loc_22D6009 ; inRdname supplied?
.text:022D6009 loc_22D6009:                               ; CODE XREF: ConstructRDName+5Cj
.text:022D6009                  push    0C9h              ; size_t
.text:022D600E                  call    _malloc
.text:022D6013                  mov     ebp, eax
.text:022D6015                  mov     ecx, 32h
.text:022D601A                  xor     eax, eax
.text:022D601C                  mov     edi, ebp
.text:022D601E                  rep stosd
.text:022D6020                  push    3                 ; size_t
.text:022D6022                  push    offset aCn_0      ; "cn="
.text:022D6027                  push    ebx               ; char *
.text:022D6028                  mov     [esp+998h+lpszDest], ebp
.text:022D602C                  stosb
.text:022D602D                  call    _strncmp
.text:022D6032                  add     esp, 10h
.text:022D6035                  test    eax, eax
.text:022D6037                  jnz     short loc_22D6063 ; rdname doesn't start with "cn="?
```

If so, the string is copied to the previously allocated heap buffer without performing any boundary checks. This may cause a heap-based buffer overflow.

```
.text:022D6039                  mov     edi, ebx
.text:022D603B                  or      ecx, 0FFFFFFFFh
.text:022D603E                  repne scasb
.text:022D6040                  not     ecx
.text:022D6042                  sub     edi, ecx
.text:022D6044                  mov     eax, ebp
.text:022D6046                  mov     edx, ecx
.text:022D6048                  mov     esi, edi
.text:022D604A                  mov     edi, ebp
.text:022D604C                  shr     ecx, 2
.text:022D604F                  rep movsd                 ; b0f!
.text:022D6051                  mov     ecx, edx
.text:022D6053                  and     ecx, 3
.text:022D6056                  rep movsb
.text:022D6058                  pop     edi
.text:022D6059                  pop     esi
.text:022D605A                  pop     ebp
.text:022D605B                  pop     ebx
.text:022D605C                  add     esp, 978h
.text:022D6062                  retn
```

If, however, the string did not start with “cn=” the function checks if “/C=” can be found anywhere in the supplied string.

```
.text:022D6063 loc_22D6063:                               ; CODE XREF: ConstructRDName+97j
.text:022D6063                  push    offset aC_0       ; "/C="
.text:022D6068                  push    ebx               ; char *
```

```
.text:022D6069                call    _strstr
.text:022D606E                mov     edi, eax
.text:022D6070                add     esp, 8
.text:022D6073                test    edi, edi
.text:022D6075                jz      loc_22D636E     ; rdname doesn't contain "/C="?
```

If “/C=” cannot be found, the string is copied into the heap buffer without performing any boundary checks. This may also lead to a heap-based buffer overflow.

```
.text:022D636E loc_22D636E:                           ; CODE XREF: ConstructRDName+D5j
.text:022D636E                mov     edi, ebx
.text:022D6370                or      ecx, 0FFFFFFFFh
.text:022D6373                xor     eax, eax
.text:022D6375                repne scasb
.text:022D6377                not     ecx
.text:022D6379                sub     edi, ecx
.text:022D637B                mov     eax, ecx
.text:022D637D                mov     esi, edi
.text:022D637F                mov     edi, ebp
.text:022D6381                shr     ecx, 2
.text:022D6384                rep movsd               ; b0f!
.text:022D6386                mov     ecx, eax
.text:022D6388                mov     eax, ebp
.text:022D638A                and     ecx, 3
.text:022D638D                rep movsb
.text:022D638F                pop     edi
.text:022D6390                pop     esi
.text:022D6391                pop     ebp
.text:022D6392                pop     ebx
.text:022D6393                add     esp, 978h
.text:022D6399                retn
```

If the string does contain “/C=”, a check is performed to see if another “/” character exists later in the string.

```
.text:022D607B                add     edi, 3
.text:022D607E                push    '/'             ; int
.text:022D6080                push    edi             ; char *
.text:022D6081                call    _strchr
.text:022D6086                add     esp, 8
.text:022D6089                test    eax, eax
.text:022D608B                jnz     short loc_22D60AE ; "/" found after "/C="?
```

If so, the whole argument passed to “/C=” is copied into a 4 byte stack buffer without performing any boundary checks. This may lead to a stack-based buffer overflow.

```
.text:022D608D                or      ecx, 0FFFFFFFFh
.text:022D6090                lea     edx, [esp+988h+szC] ; char[4]
.text:022D6094                repne scasb
.text:022D6096                not     ecx
.text:022D6098                sub     edi, ecx
.text:022D609A                mov     eax, ecx
.text:022D609C                mov     esi, edi
```

```
.text:022D609E                  mov     edi, edx
.text:022D60A0                  shr     ecx, 2
.text:022D60A3                  rep movsd               ; b0f!
.text:022D60A5                  mov     ecx, eax
.text:022D60A7                  and     ecx, 3
.text:022D60AA                  rep movsb
.text:022D60AC                  jmp     short loc_22D60BF
```

If the string does contain a "/" character after "/C=", the length of the argument to that parameter is calculated before calling strncpy() to copy the argument into the 4 byte stack buffer. As the length of the source string is used as size argument, a stack-based buffer overflow may also occur here.

```
.text:022D60AE loc_22D60AE:                             ; CODE XREF: ConstructRDName+EBj
.text:022D60AE                  sub     eax, edi
.text:022D60B0                  lea     ecx, [esp+988h+szC] ; char[4]
.text:022D60B4                  push    eax             ; size_t
.text:022D60B5                  push    edi             ; char *
.text:022D60B6                  push    ecx             ; char *
.text:022D60B7                  call    _strncpy        ; b0f!
```

Similar flawed copy operations are later performed for "/O=", "/OU=", and "/CN=" when present in the string. These may all result in stack-based buffer overflows.

Solution

The vendor has deprecated the ActiveX control, and KrCERT/CC plans to set the kill-bit.

References

RBS:	RBS-2019-004[1]
VulnDB:	202041, 202042, 202043, 202044, 202045, 202046

Timeline

2019-01-29	Vulnerability discovered.
2019-02-01	Vulnerability reported to KrCERT/CC.
2019-04-04	Alerts published to VulnDB customers.
2019-05-21	Publication of this vulnerability report.

[1] https://www.riskbasedsecurity.com/research/RBS-2019-004.pdf

About Risk Based Security

Risk Based Security offers clients fully integrated security solutions, combining real-time vulnerability and threat data, as well as the analytical resources to understand the implications of the data, resulting in not just security, but the right security.

Company History

Risk Based Security, Inc. (RBS) was established to support organizations with the technology to turn security data into actionable information and a competitive advantage. We do so by enhancing the research available and providing a first of its kind risk identification and evidence-based security management service.

As a data driven and vendor neutral organization, RBS is able to deliver focused security solutions that are timely, cost effective, and built to address the specific threats and vulnerabilities most relevant to the organizations we serve. We not only maintain vulnerability and data breach databases, we also use this information to inform our entire practice.

Solutions

VulnDB - Vulnerability intelligence, alerting, and third party library tracking based on the largest and most comprehensive vulnerability database in the world. Available as feature-rich SaaS portal or powerful API. Vendor evaluations including our Vulnerability Timeline and Exposure Metrics (VTEM), Cost of Ownership ratings, Code Maturity, and Social Risk Scores.

Cyber Risk Analytics - Extensive data breach database including interactive dashboards and breach analytics. Clients are able to gather and analyze security threat and data breach information on businesses, industries, geographies, and causes of loss. It also allows monitoring of domains for data breaches and leaked credentials as well as implementing a continuous vendor management program with our PreBreach data.

YourCISO - Revolutionary service that provides organizations an affordable security solution including policies, vulnerability scans, awareness material, incident response, and access to high quality information security resources and consulting services.

Vulnerability Assessments (VA) and Pentesting - Regularly scheduled VAs and pentests help an organization identify weaknesses before the bad guys do. Managing the most comprehensive VDB puts us in a unique position to offer comprehensive assessments, combining the latest in scanning technology and our own data. Detailed and actionable reports are provided in a clear and easy to understand language.