# SectorA01 Custom Proxy Utility Tool Analysis

threatrecon.nshc.net/2019/01/23/sectora01-custom-proxy-utility-tool-analysis/

## Overview

SectorA01 is one of the most infamous state sponsored threat actor groups globally and is unique in the sense that it is one of the only state sponsored groups with large interests in financial crime. So with the continued interest into SectorA01's financial crime activities due to the recent potential misattribution of the Ryuk ransomware [1], we decided to perform an analysis into one of the tools – a proxy utility executable – used exclusively by SectorA01 that recently caught our attention again.

Interestingly, in the Hidden Cobra FASTCash report by the US-CERT [2] in October last year, there were two versions of a "Themida packed proxy service module" (i.e. x32 and x64 versions). Our analysis of those modules showed code reuse of critical functions with the sample we are analyzing in this post, leading us to think that those samples might be an evolution of this sample.

## SectorA01 Proxy Utility

SectorA01 uses a variety of tools for different purposes, but one common custom tool used in the attacks targeting the Polish banks in 2016-2017 [3], a Taiwanese Bank in 2017 [4], and Vietnamese banks in 2018 [5] is one of their custom proxy utility executables.

The latest unique sample of this proxy utility we could find was on December 10th, 2018 from Canada. This leads us to one of a few possible theories that Canadian bank(s) may have been one of the many unreported or reported [6] targets during the time period of the attack on the Taiwanese bank based on the compilation timestamps.

As we can see from the FASTCash proxy samples below, at least one of their developers compiles the 64-bit sample immediately after compiling the 32-bit sample – behavior very normal for developers when compiling for multiple systems. The same thing can be seen for the two samples on 20 Feb 2017, and so in fact instead of calling them samples targeting a Taiwanese bank and potentially a Canadian bank, it may be more accurate to call it just one of the many pairs of 32-bit and 64-bit proxy samples produced by the group.

A proxy was also used against an unnamed Southeast Asian bank [7] which appears to be an older version of the proxy, and against an Indian bank [8] which appears to be a newer version of the proxy based our code analysis from samples in the US-CERT FASTCash report.

But despite the similarities, however, we are unable to definitively state that these samples were earlier (unnamed Southeast Asian bank) or later (FASTCash attack, such as against the Indian bank) versions of the proxy. After all, SectorA01 has more than one proxy tool in its arsenal, such as the proxy used together with their TYPEFRAME trojan [9] which has a separate code base.

| Description | Compilation Timestamp |
|---|---|
| Attack on unnamed SEA bank (old version) | 17 Sep 2014 16:59:33 |
| Attack on several Polish banks (variant) | 24 Aug 2015 10:21:52 |
| Attack on Vietnamese banks (variant) | 2 May 2016 03:24:39 |
| Attack on a Taiwanese Bank (32-bit) (variant) | 20 Feb 2017 11:09:30 |
| Sample Discovered from Canada (64-bit) (sample analyzed) | 20 Feb 2017 11:09:41 |
| FASTCash (32-bit) (new version) | 14 Aug 2017 17:14:04 |
| FASTCash (64-bit) (new version) | 14 Aug 2017 17:14:12 |

## Sample Background

This executable is a custom tunneling proxy utility tool in SectorA01's toolkit. It can be used as either a tunneling proxy server to forward traffic to another destination, or as a tunneling proxy client which requests another infected tunneling proxy server to perform requests.

Besides being used as one of several ordinary proxy servers in a chain of servers to hide the source of attacks, against one example banking target from India in the FASTCash attacks, "a proxy server was created and transactions authorized by the fake or proxy server". In this scenario, the proxy utility seems to be not used just as a secondary helper utility, but as the primary attack malware.

SectorA01 normally packs these samples with either the Themida or Enigma Protector, but in this blog post we will only be showing the analysis of the unpacked sample.

## Process Arguments

This utility requires a single process argument in order for it to run. It attempts to decode the argument and only continues its execution path if the decoded argument match the format it is expecting.

The argument is delimited by the "|" symbol, and the utility decodes up to four tokens with each token being decoded individually. The first is required and used as the primary C2 server (malware acting as tunneling proxy server) or as the URL to be requested (malware

acting as tunneling proxy client), the optional second token is used as the proxy target information, the optional third token is used as proxy server information, and the optional fourth token is an optional proxy username and password.

Each deobfuscated token is separated by a colon ":", which is used as the deobfuscated process arguments delimiter.

```
int __stdcall WinMain_0(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nShowCmd){
  //deobfuscate process arguments here
deobfuscation_complete:
  if ( strlen(deobfuscated_c2_1) != 0 && strchr(deobfuscated_c2_1, ":") ){
    …
  }
return 0;
}
```

The decoding algorithm makes use of a rotating character in an eight character string "cEzQfoPw" and the loop index to ensure that every deobfuscated character at a different index comes from a different two obfuscated characters.

We recreated this deobfuscation algorithm and created an obfuscation algorithm, which allowed us to forge our own process arguments. An example of a process argument which uses all four tokens could be "!y$t$A$s!z$S$e$U$Q$Y$1$W$U!}$d|!y#z$A$s!z$S$o$1$5$t$A$e$U!x|!y#{!}$Z$C$R$o$1$P#}$8$a!y!y|!00X!B0]0D!8#z$2$R0d$0$b!w!20c!70B0d".

| Example Token | Decoded C2 Information | Usage |
|---|---|---|
| !y$t$A$s!z$S$e$U$Q$Y$1$W$U!}$d | 192.168.1.1:443 | C2 Server (1-2 arguments)<br>Proxy Target (3-4 arguments) |
| !y#z$A$s!z$S$o$1$5$t$A$e$U!x | 172.16.1.1:443 | Proxy Target (2 arguments only) |
| !y#{!}$Z$C$R$o$1$P#}$8$a!y!y | 10.1.1.12:8080 | Proxy Server (3-4 arguments) |
| !00X!B0]0D!8#z$2$R0d$0$b!w!20c!70B0d | sector%20a01:proxy | Proxy Authentication (4 arguments only) |

Note that since the algorithm transforms every two encoded characters into one decoded character based on its character index, there are many possible two characters which will result in the same character, and finally countless different strings which would decode to a single string.

## C2 Communication

The algorithm used for C2 communications is more straightforward – a combination of ADD/XOR repeatedly from each character in a hard coded 20 character byte array "{47 B0 62 0E 69 F3 22 8D 65 40 BF 39 24 A6 C3 BB 8E 68 EB B5}" is used for decoding, and the opposite XOR/SUB repeatedly from the reversed byte array is used for encoding. The algorithm restarts for each character without context, so it essentially ends up being a character substitution table.

There are eight commands to communicate with the C2 server, encoded by either the C2 server or the proxy client then decoded by the other side. These commands are in the Russian language but as other researchers have pointed out in the past, is simply a false flag.

In fact, in one of the analyzed malware used against an unnamed Southeast Asian bank, we see that what appears to be a much earlier versions of the proxy having seven numeric-only control codes while this sample has eight Russian language control codes, with the control codes in both samples having almost the same meaning.

| Operation | Description | Hex Values over the Network |
|---|---|---|
| kliyent2podklyuchit | Malware thread created notification (client) | d1 14 23 b3 c7 b2 ac fe 70 0d 1c d1 14 b3 d7 f9 38 23 ac |
| Nachalo | Client has started (client) | 92 ab f9 38 ab 14 0d |
| ssylka | Tunneling proxy server has started (client) | c9 c9 b3 14 d1 ab |
| poluchit | Get proxy target information (server) | 70 0d 14 d7 f9 38 23 ac |
| ustanavlivat | Set proxy target information (server) | d7 c9 ac ab b2 ab 2a 14 23 2a ab ac |
| pereslat | Start a new tunneling proxy server session in new thread (server) | 70 c7 be c7 c9 14 ab ac |
| derzhat | Maintain connection (server) | 1c c7 be b6 38 ab ac |
| vykhodit | Exit (server) / Client has exited (client) | 2a b3 d1 38 0d 1c 23 ac |

## Tunneling Proxy Server

When this utility acts as a tunneling proxy server, it directly uses Windows Sockets 2 ("WS2_32") to achieve their rudimentary proxy.

```
signed __int64 __fastcall c2_ssylka(LPVOID lpThreadParameter){
  SOCKET c2Socket = begin_c2("ssylka");
  …
  SOCKET targetProxySocket = retrieveProxySocket();
  …
  start_tunnel_proxy_server(c2Socket, targetProxySocket);
  …
}

signed int __fastcall start_tunnel_proxy_server(SOCKET c2Socket, SOCKET
targetProxySocket){
  …
  numBytesReceived = recv(c2Socket, &dataToProxy, 0x2000, 0);
  …
  numBytesReceived = send(targetProxySocket, &dataToProxy, numBytesReceived, 0);
  …
}
```

## Tunneling Proxy Client

When this utility acts as a tunneling proxy client, it utilizes the more powerful embedded libcurl library (version 7.49.1 for this sample, but not always the case) to command other infected tunneling proxy servers.

```
__int64 __fastcall connect_to_proxy(__int64 fixedFunctionAddress, __int64
proxyTarget){
  …
  curl_setopt(handle, CURLOPT_URL, proxyTarget);
  …
  curl_setopt(handle, CURLOPT_PROXY, fixedFunctionAddress + 16); //refers to
deobfuscated proxy server information
  …
  curl_setopt(handle, CURLOPT_HTTPPROXYTUNNEL, 1);
  …
  if ( strlen((fixedFunctionAddress + 278)) != ) //if deobfuscated argument 4 is not
empty
    curl_setopt(handle, CURLOPT_PROXYUSERPWD); //curl_setopt argument 3 =
deobfuscated process argument 4, which is not detected by decompiler
    …
  }
  …
}
```

The CURLOPT_HTTPPROXYTUNNEL code causes the client to starts by using HTTP CONNECT to the proxy server in order to request it to forward traffic to the proxy target.

```
>Internet Protocol Version 4, Src: x.x.x.x, Dst: 10.1.1.12
>Transmission Control Protocol, Src Port: xxxxx, Dst Port: 8080, Seq: 1, Ack: 1, Len:
59
>Hypertext Transfer Protocol
  >CONNECT 192.168.1.1:443 HTTP/1.1\r\n
    >[Expert Info (Chat/Sequence): CONNECT 192.168.1.1:443 HTTP/1.1\r\n]
      Request Method: CONNECT
      Request URI: 192.168.1.1:443
      Request Version: HTTP/1.1
    Host: 192.168.1.1:443\r\n
```

## The FASTCash Connection

In October last year, the US-CERT reported about the “FASTCash” campaign by SectorA01, which was essentially an ATM cash-out scheme whereby SectorA01 remotely compromised bank payment switch applications to simultaneously physically withdraw from ATMs in many countries and steal millions of dollars.

Some of the artifacts used in the campaign included proxy modules, a RAT, and an installer application. When we performed a preliminary analysis and compared the FASTCash proxy module to the proxy module analyzed in this post, we found algorithmic similarities between the decoding/encoding functions, the process argument deobfuscation function, and the proxy function.

However, the FASTCash proxy module also had more functions in them with new capabilities as described briefly in the US-CERT FASTCash Malware Analysis Report [10]. Additionally, our own analysis showed that they have also updated the use of amateur-ish strings which were previously easily detectable from memory and obviously malicious, to now hiding or removing those custom strings. This is their normal behavior as it has been known that they are constantly modifying their own source code, and these similarities and developments leads us to think that the FASTCash proxy module might be an evolution of their previous proxy module.

## Summary

Attribution is a complex and controversial topic, but regardless, correctly attributing a threat to a particular threat group is a far easier task than correctly attributing the threat to or being linked to a particular nation state. Given even a single piece of complex enough custom malware believed to be in possession by only a single group and context behind the attack, it is possible to have some degree of confidence of which group was behind the attack.

But even custom malware source code can get stolen, the executable itself repackaged, or the functions recreated. In a simpler scenario, false flags such as strings and metadata could also be placed.

Regarding the initial attribution of the Ryuk ransomware, however, while others have focused on the misattribution, our view is that even if it was correct it would simply have been a lucky guess. Basing attribution solely on the usage of a single privately purchasable malware is fundamentally flawed, and the simple truth is that no organization in the world would be able to track every piece of malware to know what is being sold in the dark and deep web anyway.

That is why in order to have a higher degree of confidence of who is behind an attack, the entirety of the threat's tactics, techniques, and procedures (TTPs) need to be analyzed across multiple events using both trusted public and vetted private sources.

SectorA01 shows no signs of stopping their attacks against financial sectors worldwide and although they have been constantly modifying their code protectors, functions, and algorithms, there will be traces of similarities across different versions of their tools. Our Threat Recon Team will continue tracking such events and malware and report on our findings.

## Indicators of Compromise (IoCs)

**Unpacked Sample (SHA-256)**
0d75d429c1cc3550b2961be84af777f8bed287a44a144b7a47988c601e1e9a27

**Memory Dump Samples from US-CERT FASTCash Report (SHA-256)**
9ddacbcd0700dc4b9babcd09ac1cebe23a0035099cb612e6c85ff4dffd087a26
1f2cd2bc23556fb84a51467fedb89cbde7a5883f49e3cfd75a241a6f08a42d6d

**Packed Sample from Polish banks attack (SHA-256)**
d4616f9706403a0d5a2f9a8726230a4693e4c95c58df5c753ccc684f1d3542e2

**Sample from Taiwanese bank attack (SHA-256)**
9a776b895e93926e2a758c09e341accb9333edc1243d216a5e53f47c6043c852

**Sample from Vietnamese banks attack (SHA-256)**
f3ca8f15ca582dd486bd78fd57c2f4d7b958163542561606bebd250c827022de

**Attack on Unnamed SEA Bank (TCP Tunnel Tool) (SHA-256)**
19bba0a7669a0109a6d2184bc0135ea4581449c8f5f0ef8a04af057447635cab

## References

[1] Ryuk Ransomware Attack: Rush to Attribution Misses the Point

[2] HIDDEN COBRA – FASTCash Campaign

[3] Włamania do kilku banków skutkiem poważnego ataku na polski sektor finansowy

[4] TAIWAN HEIST: LAZARUS TOOLS AND RANSOMWARE

[5] High alert against malicious code attacks in Vietnam

[6] BMO and CIBC-owned Simplii Financial reveal hacks of customer data

[7] LAZARUS UNDER THE HOOD

[8] North Korean connection to Cosmos hacking? Signs point to Bangladesh heist masterminds

[9] MAR-10135536-12 – North Korean Trojan: TYPEFRAME

[10] MAR-10201537 – HIDDEN COBRA FASTCash-Related Malware