Have Your Cake and Eat it Too? An Overview of UNC2891

mandiant.com/resources/unc2891-overview

The Mandiant Advanced Practices team previously published a threat research blog post that provided an overview of UNC1945 operations where the actor compromised managed services providers to gain access to targets in the financial and professional consulting industries.

Since that time, Mandiant has investigated and attributed several intrusions to a threat cluster we believe has a nexus to this actor, currently being tracked as UNC2891. Through these investigations, Mandiant has discovered additional techniques, malware, and utilities being used by UNC2891 alongside those previously observed in use by UNC1945. Despite having identified significant overlaps between these threat clusters, Mandiant has not determined they are attributable to the same actor.

- UNC2891 intrusions appear to be financially motivated and in some cases spanned several years through which the actor had remained largely undetected.
- UNC2891 demonstrated fluency and expertise in Unix and Linux environments, mostly through the targeting of Oracle Solaris based systems with TINYSHELL and SLAPSTICK backdoors.
- Mandiant observed UNC2891 operate with a high degree of OPSEC and leverage both public and private malware, utilities, and scripts to remove evidence and hinder response efforts.
- Mandiant discovered a previously unknown rootkit for Oracle Solaris systems that UNC2891 used to remain hidden in victim networks, we have named this CAKETAP.
- One Variant of CAKETAP manipulated messages transiting a victims Automatic Teller Machine (ATM) switching network. It is believed this was leveraged as part of a larger operation to perform unauthorized cash withdrawals at several banks using fraudulent bank cards.

Extensive Use of SLAPSTICK and TINYSHELL Backdoors

Like past UNC1945 intrusions, Mandiant observed UNC2891 make extensive use of the Pluggable Authentication Module (PAM) based backdoor we track as SLAPSTICK to aid with credential harvesting, and to provide backdoor access to compromised machines in victim networks. As detailed in our previous blog post, SLAPSTICK provides persistent backdoor access to infected systems with a hard-coded *magical password*, it also logs authentication attempts and corresponding passwords in an encrypted log file. Although this is expected to have tremendously assisted UNC2891 with credential harvesting and lateral movement activities, it also provided valuable information to Mandiant Incident Responders. Although SLAPSTICK log files were often timestomped, Mandiant was able to decode them and trace some of the actor's lateral movement activities through the usage of the backdoor provided *magical password*.

2021 Jan 16 14:10:06	/usr/sbin/sshd	sshd	user1	plaintextpassword	server1	Authentication failure
2021 Jan 18 22:50:45	/usr/sbin/sshd	sshd	root	rBa4JZpFABIHj67rWONnk29	172.16.10.10	Magical password
2021 Jan 19 09:27:11	/usr/sbin/sshd	sshd	user2	plaintextpassword	server2	Authentication failure
2021 Feb 02 05:41:43	/usr/bin/passwd	passwd	root	plaintextpassword		Success
2021 Feb 04 21:03:32	/usr/sbin/sshd	sshd	user3	plaintextpassword	10.10.10.101	Authentication failure
2021 Mar 22 20:15:06	/usr/sbin/sshd	sshd	user4	plaintextpassword	10.10.10.102	Success
2021 Mar 23 18:19:12	/usr/sbin/sshd	sshd	user5	plaintextpassword	server3	Authentication failure

Figure 1: Example SLAPSTICK decoded log (fabricated)

Alongside SLAPSTICK, UNC2891 often installed a custom variant of the publicly available TINYSHELL backdoor. UNC2891 TINYSHELL backdoors leveraged an external encrypted configuration file and some variants included additional functionality, such as the ability to communicate via a HTTP proxy with basic authentication. In line with the group's familiarity with Unix and Linux based systems, UNC2891 often named and configured their TINYSHELL backdoors with values that masqueraded as legitimate services that might be overlooked by investigators, such as systemd (SYSTEMD), name service cache daemon (NCSD), and the Linux at daemon (ATD).

TINYSHELL Backdoor File Paths	TINYSHELL Configuration File Paths
/usr/lib/libhelpx.so.1	/usr/lib/libatdcf.so
/usr/lib/systemd/systemd-helper	/usr/lib/libnscd.so.1
/usr/sbin/nscd	/usr/lib/libsystemdcf.so
	/var/ntp/ntpstats/1

Table 1: Observed TINYSHELL file paths

Example Decoded configuration

```
pm_systemd_mag <32-character string>
systemd_nme <system id>
pm_systemd_adr <C2 IP address/domain>
pm_systemd_prt <443 or 53>
pm_systemd_tme 300
systemd_non1 none
systemd_non2 none
systemd_non3 none
systemd_non4 none
```

Table 2: Example decoded TINYSHELL configuration (systemd variant)

In the case of the systemd variant, UNC2891 also leveraged systemd service unit files for persistence of the TINYSHELL backdoor.

/usr/lib/systemd/system/systemd-helper.service

```
[Unit]
Description=Rebuild Hardware Database
[Service]
Type=forking
ExecStart=/lib/systemd/systemd-helper
[Install]
WantedBy=multi-user.target
```

Table 3: Service unit file used for TINYSHELL persistence

Based on analyzed configurations, UNC2891 had configured TINYSHELL backdoors in a multi-hop structure that leveraged several compromised internal servers for command and control. In one case, Mandiant found evidence that suggests the actor had chained different TINYSHELL variants together to obtain remote access to a server inside a network segment with network restrictions.

To keep their network of TINYSHELL connections hidden, UNC2891 had installed and configured a rootkit to filter out these connections from network connection related APIs (keep reading for details on the CAKETAP rootkit). UNC2891 configured remotely accessible systems with TINYSHELL backdoors that used dynamic DNS domains for their external command and control channel. These domains were created per-host and were not used more than once, the subdomains sometimes resembled the hostname of the compromised machine. Mandiant was unable to collect passive DNS data for these dynamic DNS domains, suggesting that UNC2891 had likely enabled IP resolution for short periods of time when access to the network was required. At one victim, these TINYSHELL backdoors were configured to perform communications using TCP over port 53 and 443, likely as a mechanism to bypass outbound network protections, blend in with existing traffic, and evade detection.

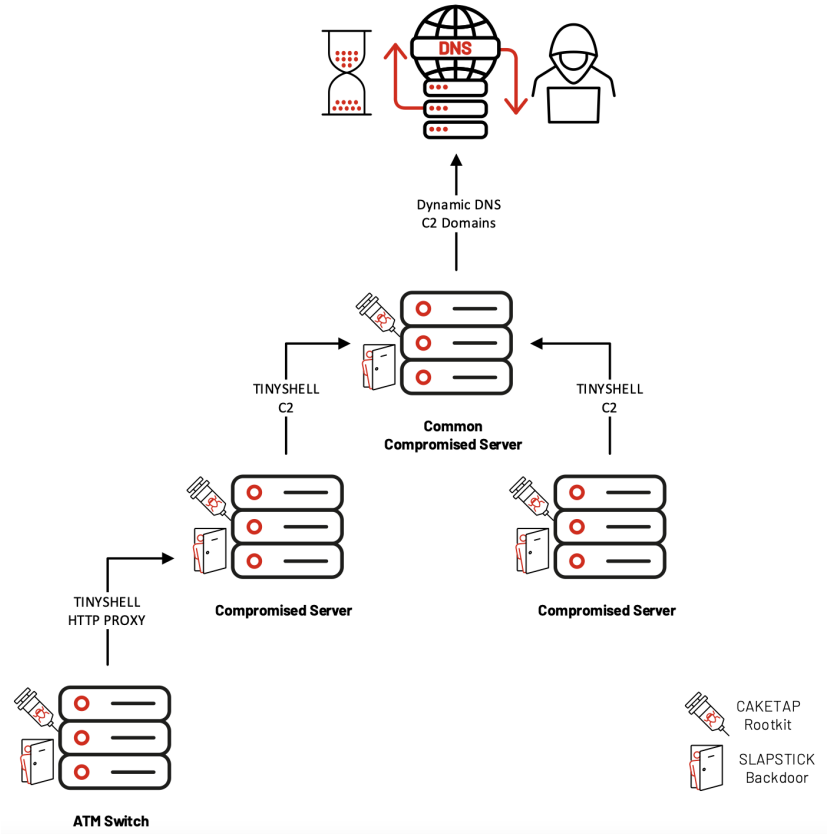

Figure 2: Example of TINYSHELL command and control used by UNC2891

STEELHOUND, STEELCORGI and Environment Variable Keying

UNC2891 often made use of the STEELCORGI in-memory dropper which decrypts its embedded payloads by deriving a ChaCha20 key from the value of an environment variable obtained at runtime. In many cases, Mandiant was unable to recover the requisite environment variables to decrypt the embedded payloads. However, in the limited samples we were able to decrypt, UNC2891 had deployed different versions of an extensive toolkit which appears to be developed under the name SUN4ME. SUN4ME contains tools for network reconnaissance, host enumeration, exploitation of known vulnerabilities, log wiping, file operations, as well as common shell utilities. Yoroi has previously published information about this toolkit following our previous blog post on UNC1945's usage of STEELCORGI.

Mandiant discovered UNC2891 leveraging a similar in-memory dropper that also used environment variables to decrypt its embedded payload but instead relied on RC4 encryption, we have named this STEELHOUND. In addition to functioning as dropper for an embedded payload, STEELHOUND is also able to encrypt new payloads by encrypting a target binary and writing it to disk along with a copy of itself and an end-of-file configuration.

WINGHOOK and WINGCRACK

During these investigations, Mandiant also discovered a family of keylogger malware we have named WINGHOOK and WINGCRACK.

- WINGHOOK is a keylogger for Linux and Unix based operating systems. It is packaged as a shared library (SO file) that hooks the *read* and *fgets* functions, which are two common functions used for processing user input. The captured data is stored in an encoded format in the directory */var/tmp/* with a filename that begins with *.zmanDw*.
- WINGCRACK is a utility that can decode and display the content of files containing encoded keylog data from WINGHOOK. The malware author appears to refer to these encoded files as "schwing" files.

Utilities Observed

Mandiant previously observed UNC1945 use a large amount of different public and private tools during their intrusions, and this was also true for UNC2891. Mandiant discovered additional utilities that were leveraged by UNC2891:

- BINBASH is a simple ELF utility that executes a shell after setting the group ID and user ID to either "root" or specified values. BINBASH appears to be a compilation of the source code.
- WIPERIGHT is an ELF utility that clears specific log entries on Linux and Unix based systems. It can remove entries associated with a given user in the lastlog, utmp/utmpx, wtmp/wtmpx, and pacct logs. It appears to have originated from available source code, and possibly a more recent version.

- MIGLOGCLEANER is another ELF utility that wipes logs or remove certain strings from logs on Linux and Unix based systems. It is publicly available on GitHub.

Whilst seemingly uncommon amongst threat actors, UNC2891 frequently used the uuencoding scheme to encode and decode files, such as malware binaries or files containing output from extensive host enumeration scripts. The actor often leveraged simple Perl wrapper scripts that performed uuencoding and uudecoding functions.

CAKETAP

CAKETAP is a kernel module rootkit that UNC2891 deployed on key server infrastructure running Oracle Solaris. CAKETAP can hide network connections, processes, and files. During initialization, it removes itself from the loaded modules list and updates the *last_module_id* with the previously loaded module to hide its presence.

A hook is installed into the function *ipcl_get_next_conn*, as well as several functions in the *ip* module. This enables CAKETAP to filter out any connections that match an actor-configured IP address or port (local or remote).

One way to identify CAKETAP running on a Solaris system is to check for the presence of this hook. The following shows an example command to identify a hooked *ipcl_get_next_conn* function (Note: The *mdb* command may require special permissions on the system):

```
root@solaris:~# echo 'ipcl_get_next_conn::dis -n 0 ; ::quit' | mdb -k
```

The output in a clean SPARC Solaris system would look similar to the following:

```
ipcl_get_next_conn: save %sp, -0xb0, %sp
```

A hooked function would begin with the *sethi* instruction as follows (the constant *0x11971c00* will change from instance to instance depending on where CAKETAP is loaded):

```
ipcl_get_next_conn: sethi %hi(0x11971c00), %g1
```

Additional hooks are installed into the *mkdirat* (make directory at) and *getdents64* (get directory entries) system calls. CAKETAP uses the *mkdirat* hook to receive commands from paths containing the signal string. Commands include configuring network filters, display or update its configuration, and to unhide itself. The *getdents64* hook enables CAKETAP to hide files or directories on the file system containing the secret signal string. Table 4 contains the signal strings for the CAKETAP hooks.

Secret	Usage
.caahGss187	mkdirat hook signal string
.zaahGss187	getdents64 hook signal string

Table 4: Observed secrets for CAKETAP hooks

The *mkdirat* hook enabled UNC2891 to control and configure CAKETAP through existing backdoor access to compromised servers by issuing shell commands that leverage these system calls (e.g. mkdir for mkdirat). A single character appended to the signal string indicated which command was to be executed. The following commands were observed:

Command	Function
Empty	Add the CAKETAP module back to loaded modules list
M	Change the signal string for the *getdents64* hook
I	Add a network filter (format <IP>p<PORT>)
i	Remove a network filter
P	Set the current thread TTY to not be filtered by the *getdents64* hook

p	Set all TTYs to be filtered by the *getdents64* hook
S	Displays the current configuration

Table 5: Observed CAKETAP commands

For example, to configure a new network filter and display the current configuration, the following commands might be used:

- *mkdir /some/path/.caahGss187I192.168.1.10p80* - Add network filter for 192.168.1.10:80
- *mkdir /some/path/.caahGss187S* - Display current configuration

The hook installed into *getdents64* filtered output to hide presence of the signal string in directory contents.

Mandiant observed UNC2891 load CAKETAP with the module name *ipstat* from attacker created directories that often resided somewhere inside the */var* directory tree.

CAKETAP Unauthorized Transactions

Memory forensics from one victim's ATM switch server revealed a variant of CAKETAP with additional network hooking functionality that intercepted specific messages relating to card and pin verification. Evidence suggests that this variant of CAKETAP was used as part of an operation to perform unauthorized transactions using fraudulent bank cards.

This CAKETAP variant targeted specific messages destined for the Payment Hardware Security Module (HSM). This additional network hooking performed several functions:

1. Manipulation of card verification messages:
   CAKETAP altered the mode of certain outgoing messages to disable card verification. This resulted in the HSM not performing the proper card verification and instead generating a valid response. Fraudulent bank cards generated verification messages using a custom algorithm using the Primary Account Number (PAN) and other parameters which served as a "marker" for CAKETAP. CAKETAP examined outgoing messages and if it matched the algorithm, CAKETAP identified the card as fraudulent and stored the PAN in memory to use in the following step.
2. Replay of PIN verification messages:
   CAKETAP examined outgoing PIN verification messages that matched certain conditions and identified those with a Primary Account Number (PAN) that reflected a fraudulent card. If the message was not for a fraudulent card, it would save the message internally and send it unmodified, as to not interrupt legitimate ATM PIN verifications. However, if it was for a fraudulent card, CAKETAP would instead replace the message content with data from a previously saved message. This was effectively a replay attack that resulted in a bypass of PIN verification for fraudulent cards.

Based on Mandiant's investigation findings, we believe that CAKETAP was leveraged by UNC2891 as part of a larger operation to successfully use fraudulent bank cards to perform unauthorized cash withdrawals from ATM terminals at several banks.

Conclusion

UNC2891 maintains a high level of OPSEC and employs several techniques to evade detection. The actor uses their skill and experience to take full advantage of the decreased visibility and security measures that are often present in Unix and Linux environments. Mandiant expects that UNC2891 will continue to capitalize on this and perform similar operations for financial gain that target mission critical systems running these operating systems.

While some of the overlaps between UNC2891 and UNC1945 are notable, it is not conclusive enough to attribute the intrusions to a single threat group. For example, it is possible that significant portions of UNC2891 and UNC1945 activity are carried out by an entity that is a common resource to multiple threat actors, which could explain the perceived difference in intrusion objectives—a common malware developer or an intrusion partner, for example. Regardless, Mandiant is releasing this information on the actor to raise awareness of the fraudulent activity and aid defenders in uncovering further UNC2891 operations.

YARA

The following YARA rules are not intended to be used on production systems or to inform blocking rules without first being validated through an organization's own internal testing processes to ensure appropriate performance and limit the risk of false positives. These rules are intended to serve as a starting point for hunting efforts to identify samples, however, they may need adjustment over time if the malware family changes.

```
rule TINYSHELL
{
    meta:
        author = "Mandiant "
    strings:
        $sb1 = { C6 00 48 C6 4? ?? 49 C6 4? ?? 49 C6 4? ?? 4C C6 4? ?? 53 C6 4? ?? 45 C6 4? ?? 54 C6 4? ?? 3D C6 4? ?? 46 C6 4? ?? 00 }
        $sb2 = { C6 00 54 C6 4? ?? 4D C6 4? ?? 45 C6 4? ?? 3D C6 4? ?? 52 }
        $ss1 = "fork" ascii fullword wide
        $ss2 = "socket" ascii fullword wide
        $ss3 = "bind" ascii fullword wide
        $ss4 = "listen" ascii fullword wide
        $ss5 = "accept" ascii fullword wide
        $ss6 = "alarm" ascii fullword wide
        $ss7 = "shutdown" ascii fullword wide
        $ss8 = "creat" ascii fullword wide
        $ss9 = "write" ascii fullword wide
        $ss10 = "open" ascii fullword wide
        $ss11 = "read" ascii fullword wide
        $ss12 = "execl" ascii fullword wide
        $ss13 = "gethostbyname" ascii fullword wide
        $ss14 = "connect" ascii fullword wide
    condition:
        uint32(0) == 0x464c457f and 1 of ($sb*) and 10 of ($ss*)
}


rule TINYSHELL_SPARC
{
    meta:
        author = "Mandiant"
    strings:
        $sb_xor_1 = { DA 0A 80 0C 82 18 40 0D C2 2A 00 0B 96 02 E0 01 98 03 20 01 82 1B 20 04 80 A0 00 01 82 60 20 00 98 0B 00 01 C2 4A 00 0B 80 A0 60 00 32 BF FF F5 C2 0A 00 0B 81 C3 E0 08 }
        $sb_xor_2 = { C6 4A 00 00 80 A0 E0 00 02 40 00 0B C8 0A 00 00 85 38 60 00 C4 09 40 02 84 18 80 04 C4 2A 00 00 82 00 60 01 80 A0 60 04 83 64 60 00 10 6F FF F5 90 02 20 01 81 C3 E0 08 }
    condition:
        uint32(0) == 0x464C457F and (uint16(0x10) & 0x0200 == 0x0200) and (uint16(0x12) & 0x0200 == 0x0200) and 1 of them
}
```

```
rule SLAPSTICK
{
    meta:
        author = "Mandiant "
    strings:
        $ss1 = "%Y %b %d %H:%M:%S    \x00"
        $ss2 = "%-23s %-23s %-23s\x00"
        $ss3 = "%-23s %-23s %-23s %-23s %-23s %s\x0a\x00"
    condition:
        (uint32(0) == 0x464c457f) and all of them
}


rule STEELCORGI
{
    meta:
        author = "Mandiant "
    strings:
        $s1 = "\x00\xff/\xffp\xffr\xffo\xffc\xff/\xffs\xffe\xffl\xfff\xff/\xffe\xffx\xffe\x00"
        $s2 =
"\x00\xff/\xffv\xffa\xffr\xff/\xffl\xffi\xffb\xff/\xffd\xffb\xffu\xffs\xff/\xffm\xffa\xffc\xffh\xffi\xffn\xffe\xff-
\xffi\xffd\x00"
        $sb1 = { FE 1B 7A DE 23 D1 E9 A1 1D 7F 9E C1 FD A4 }
        $sb2 = { 3B 8D 4F 45 7C 4F 6A 6C D8 2F 1F B2 19 C4 45 6A 6A }
    condition:
        (uint32(0) == 0x464c457f) and all of them
}
```

Indicators of Compromise

Malware Family	MD5	SHA1	SHA256
STEELCORGI	e5791e4d2b479ff1dfee983ca6221a53	e55514b83135c5804786fa6056c88988ea70e360	95964d669250f0ed161409b93f
STEELCORGI	0845835e18a3ed4057498250d30a11b1	c28366c3f29226cb2677d391d41e83f9c690caf7	7d587a5f6f36a74dcfbcbaecb2
STEELCORGI	d985de52b69b60aa08893185029bcb31	a3e75e2f700e449ebb62962b28b7c230790dc25d	cd06246aff527263e409dd779b
TINYSHELL	4ff6647c44b0417c80974b806b1fbcc3	fa36f10407ed5a6858bd1475d88dd35927492f52	55397addbea8e5efb8e6493f3b
TINYSHELL	13f6601567523e6a37f131ef2ac4390b	4228d71c042d08840089895bfa6bd594b5299a89	24f459a2752175449939037d6a
TINYSHELL	4e9967558cd042cac8b12f378db14259	018bfe5b9f34108424dd63365a14ab005e249fdd	5f46a25473b9dda834519093c6
STEELHOUND	a4617c9a4bde94e867f063c28d763766	097d3a15510c48cdb738344bdf00082e546827e8	161a2832baba6ff6f9f1b52ed8

MITRE ATT&CK

- Discovery:
  - T1016:System Network Configuration Discovery
  - T1018:Remote System Discovery
  - T1049:System Network Connections Discovery
  - T1082:System Information Discovery
  - T1083:File and Directory Discovery
  - T1135:Network Share Discovery
- Lateral Movement:
  - T1021:Remote Services
  - T1021.004:SSH
- Credential Access:
  - T1003:OS Credential Dumping
  - T1003.008:/etc/passwd and /etc/shadow
  - T1110:Brute Force
  - T1110.001:Password Guessing
  - T1552:Unsecured Credentials
  - T1552.003:Bash History
  - T1552.004:Private Keys
  - T1556.003:Pluggable Authentication Modules
- Command and Control:
  - T1090:Proxy
  - T1095:Non-Application Layer Protocol
  - T1105:Ingress Tool Transfer
  - T1572:Protocol Tunneling
  - T1573.001:Symmetric Cryptography
- Execution:
  - T1053.001:At (Linux)
  - T1059:Command and Scripting Interpreter
  - T1059.004:Unix Shell
- Collection:
  - T1056.001:Keylogging
  - T1560:Archive Collected Data
  - T1560.001:Archive via Utility
  - T1560.002:Archive via Library
- Defense Evasion:
  - T1014:Rootkit
  - T1027:Obfuscated Files or Information
  - T1070:Indicator Removal on Host
  - T1070.002:Clear Linux or Mac System Logs
  - T1070.004:File Deletion
  - T1070.006:Timestomp
  - T1140:Deobfuscate/Decode Files or Information
  - T1480.001:Environmental Keying
  - T1548.001:Setuid and Setgid
  - T1620:Reflective Code Loading
- Persistence:
  - T1543.002:Systemd Service
  - T1547.006:Kernel Modules and Extensions