P2Pinfect - New Variant Targets MIPS Devices

cadosecurity.com/p2pinfect-new-variant-targets-mips-devices/

December 4, 2023

Blog

December 4, 2023

Summary

- A new P2Pinfect variant compiled for the Microprocessor without Interlocked Pipelined Stages (MIPS) architecture has been discovered
- This demonstrates increased targeting of routers, Internet of Things (IoT) and other embedded devices by those behind P2Pinfect
- The new sample includes updated evasion mechanisms, making it more difficult for researchers to dynamically analyse
- These include Virtual Machine (VM) detection methods for embedded payloads, along with debugger detection and anti-forensics on Linux hosts

Introduction

Since July 2023, Cado Security Labs have been monitoring and reporting on the rapid growth of a cross-platform botnet, named “P2Pinfect”. As the name suggests, the malware – written in Rust – acts as a botnet agent, connecting infected hosts in a peer-to-peer topology. In early samples, the malware exploited Redis for initial access – a relatively common technique in cloud environments.

There are a number of methods for exploiting Redis servers, several of which appear to be utilised by P2Pinfect. These include exploitation of CVE-2022-0543 – a sandbox escape vulnerability in the LUA scripting language (reported by Unit42), and, as reported previously

by Cado Security Labs, an unauthorised replication attack resulting in the loading of a malicious Redis module.

Cado Security Labs researchers have since encountered a new variant of the malware, specifically targeting embedded devices based on 32-bit MIPS processors, and attempting to bruteforce SSH access to these devices. It's highly likely that by targeting MIPS, the P2Pinfect developers intend to infect routers and IoT devices with the malware. Use of MIPS processors is common for embedded devices and the architecture has been previously targeted by botnet malware, including high-profile families like Mirai, and its variants/derivatives.

Not only is this an interesting development in that it demonstrates a widening of scope for the developers behind P2Pinfect (more supported processor architectures equals more nodes in the botnet itself), but the MIPS32 sample includes some notable defence evasion techniques.

This, combined with the malware's utilisation of Rust (aiding cross-platform development) and rapid growth of the botnet itself, reinforces previous suggestions that this campaign is being conducted by a sophisticated threat actor.

Initial Access

Cado researchers encountered the MIPS variant of P2Pinfect after triaging files uploaded via SFTP and SCP to a SSH honeypot. Although earlier variants had been observed scanning for SSH servers, and attempting to propagate the malware via SSH as part of its worming procedure, Cado Security Labs had yet to observe successful implantation of a P2Pinfect sample using this method – until now.

In keeping with similar botnet families, P2Pinfect includes a number of common username/password pairs embedded within the MIPS binary itself. The malware will then iterate through these pairs, initiating a SSH connection with servers identified during the scanning phase to conduct a brute force attack.

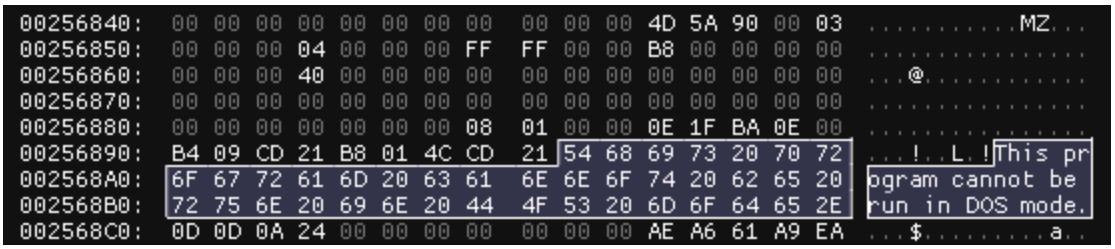
It was assumed that SSH would be the primary method of propagation for the MIPS variant, due to routers and other embedded devices being more likely to utilise SSH. However, additional research shows that it is in fact possible to run the Redis server on MIPS. This is achievable via an OpenWRT package named redis-server.

It's unclear what use-case running Redis on an embedded MIPS device solves, or whether it's commonly encountered in the wild. If such a device is compromised by P2Pinfect and has the redis-server package installed, it's perfectly feasible for that node to then be used to compromise new peers via one of the reported P2Pinfect attack patterns, involving exploitation of Redis or SSH bruteforcing.

Static Analysis

The MIPS variant of P2Pinfect is a 32-bit, statically-linked, ELF binary with stripped debug information. Basic static analysis revealed the presence of an additional ELF executable, along with a 32-bit Windows DLL in the PE32 format – more on this later.

This piqued the interest of Cado analysts, as it's unusual to encounter a compiled ELF with an embedded DLL. Consequently, it was a defining feature of the original P2Pinfect samples.



Embedded Windows PE32 executable

Further analysis of the host executable revealed a structure named "BotnetConf" with members consistent in naming with the original P2Pinfect samples.

```
{
    "i": 3,
    "c": 1696145828,
    "e": 1704038399,
    "t": {
        "C": {
            "fast_exp_syn_scan_limit": 4000,
            "slow_exp_syn_scan_limit": 4000,
            "fast_exp_limit": 400,
            "slow_exp_limit": 400,
            "local_exp_limit": 15,
            "local_exp_delay": 604800,
            "ping_delay_secs": 60,
            "file_servers_online_check_delay_secs": 180,
            "a_ping_main_delay_secs": 90,
            "m_ping_main_delay_secs": 160,
            "max_load_addrs_len": 30,
            "update_check_delay_secs": 3600,
            "a_main_file_sync_delay_secs": 1800,
            "a_all_file_sync_delay_secs": 3600,
            "m_main_file_sync_delay_secs": 7200,
            "m_all_file_sync_delay_secs": 10800,
            "ssh_too_slow": null,
            "ssh_auth_timeout": null,
            "ssh_dict": null,
            "redis_dict": null,
            "local_net_ssh_dict": null,
            "local_net_redis_dict": null,
            "enable_kill": null
        }
    }
},
```


Example of a partially-populated version of the BotnetConf struct

As the name suggests, this structure defines the configuration of the malware itself, whilst also storing the IP addresses of nodes identified during the SSH and Redis scans. This, in combination with the embedded ELF and DLL, along with the use of the Rust programming language allowed us to positively attribute this sample to the P2Pinfect family.

Updated Evasion – consulting TracerPid

One of the more interesting aspects of the MIPS sample was the inclusion of a new evasion technique. Shortly after execution, the sample calls `fork()` to spawn a child process.

The child process then proceeds to access `/proc` using `openat()`, determines its own Process Identifier (PID) using the Linux `getpid()` syscall, and then uses this PID to consult the relevant `/proc` subdirectory and read the `status` file within that. Note that this is likely achieved in the source code by resolving the symbolic link at `/proc/self/status`.

```
Name:   bioset
State:  S (sleeping)
Tgid:   852
Ngid:   0
Pid:    852
PPid:   2
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 32
Groups:
Threads:        1
SigQ:   0/1942
SigPnd: 0000000000000000000000000000000
ShdPnd: 0000000000000000000000000000000
SigBlk: 0000000000000000000000000000000
SigIgn: ffffffffffffffffffffffffffffffff
SigCgt: 0000000000000000000000000000000
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
Cpus_allowed:   1
Cpus_allowed_list:      0
voluntary_ctxt_switches:        2
nonvoluntary_ctxt_switches:     0
```

Example contents of /proc/pid/status when process not being traced

`/proc/<pid>/status` contains human-readable metadata and other information about the process itself, including memory usage and the name of the command currently being run. Importantly, the `status` file also contains a field `TracerPID:`. This field is assigned a value of 0 if the current process is not being traced by dynamic analysis tools, such as `strace` and `ltrace`.

```
loc_4F58B0:                             # CODE XREF: mw_evasion_func+198↑j
                addiu   $v0, $sp, 0x370+var_B0
                addiu   $v1, $sp, 0x370+var_290
                addiu   $a0, $sp, 0x370+var_228
                li      $a1, 0
                lw      $at, (dword_700550 - 0x7084E0)($fp)
                addiu   $a2, $at, (aStatus - 0x6E0000)  # # "status"
                li      $a3, 6

loc_4F58CC:                             # CODE XREF: mw_evasion_func+1D8↓j
                beq     $a1, $a3, loc_4F58EC
                nop
                addu    $at, $a2, $a1
                lbu     $at, 0($at)
                bnez    $at, loc_4F58CC
                addiu   $a1, 1
                break
                break
```

Example MIPS disassembly showing reading of /proc/pid/status file

If this value is non-zero, the MIPS variant of P2Pinfect determines that it's being analysed and will immediately terminate both the child process and its parent.

```
read(5, "Name:\tmips_embedded_p\nUmask:\t002", 32) = 32
read(5, "2\nState:\tR (running)\nTgid:\t975\nN", 32) = 32
read(5,
"gid:\t0\nPid:\t975\nPPid:\t1\nTracerPid:\t971\nUid:\t0\t0\t0\t0\nGid:\t0\t0\t0\t0",
64) = 64
read(5, "\nFDSize:\t32\nGroups:\t0
\nNStgid:\t975\nNSpid:\t975\nNSpgid:\t975\nNSsid:\t975\nVmPeak:\t    3200
kB\nVmSize:\t    3192 kB\nVmLck:\t       0 kB\n", 128) = 128
read(5, "VmPin:\t       0 kB\nVmHWM:\t    1564 kB\nVmRSS:\t    1560 kB\nRssAnon:\t
60 kB\nRssFile:\t    1500 kB\nRssShmem:\t       0 kB\nVmData:\t     108 kB\nVmStk:\t
132 kB\nVmExe:\t    2932 kB\nVmLib:\t       8 kB\nVmPTE:\t      16 kB\nVmSwap:\t
0 kB\nCoreDumping:\t0\nThre", 256) = 256
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x77ff1000
read(5,
"ads:\t1\nSigQ:\t0/1749\nSigPnd:\t0000000000000000000000000000000\nShdPnd:\t00000000
0000000000000000000000\nSigBlk:\t0000000000000000000000000000000\nSigIgn:\t0000000
0000000000000001000\nSigCgt:\t0000000000000000000000000000600\nCapInh:\t000000
0000000000\nCapPrm:\t0000003fffffffff\nCapEff:\t0000003fffffffff\nCapBnd:\t0000003fff
ffffff\nCapAmb:\t0000000000000000\nNoNewPrivs:\t0\nSeccomp:\t0\nSpeculation_Store_Byp
ass:\tunknown\nCpus_allowed:\t1\nCpus_allowed_list:\t0\nMems_allowed:\t1\nMems_allowe
d_list:\t0\nvoluntary_ctxt_switches:\t92\nn", 512) = 512
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x77fef000
munmap(0x77ff1000, 4096)                = 0
read(5, "onvoluntary_ctxt_switches:\t0\n", 1024) = 29
read(5, "", 995)                        = 0
close(5)                                = 0
munmap(0x77fef000, 8192)                = 0
sigaltstack({ss_sp=NULL, ss_flags=SS_DISABLE, ss_size=8192}, NULL) = 0
munmap(0x77ff4000, 12288)               = 0
exit_group(-101)                        = ?
+++ exited with 155 +++
```

Strace output demonstrating TracerPid evasion technique

Updated Evasion – disabling core dumps

Interestingly, the sample will also attempt to disable Linux core dumps. This is likely used as an anti-forensics procedure as the memory regions written to disk as part of the core dump can often contain internal information about the malware itself. In the case of P2Pinfect, this would likely include information such as IP addresses of connected peers and the populated BotnetConf struct mentioned previously.

It’s also possible that the sample prevents core dumps from being created to protect the availability of the MIPS device itself. Low-powered embedded devices are unlikely to have lots of local storage available to them and core dumps could quickly fill what little storage they do have, affecting performance of the device itself.

```
loc_4F5874:                             # CODE XREF: mw_evasion_func+164↑j
                lw      $at, (off_700564 - 0x7084E0)($fp)
                addiu   $t9, $at, (sub_5E41EC - 0x5E0000)
                jalr    $t9 ; sub_5E41EC
                nop
                move    $a0, $v0
                lw      $at, (dword_700550 - 0x7084E0)($fp)
                addiu   $a1, $at, (aFailedToDisabl - 0x6E0000)  # "Failed to disable core-dumps via rlimit"
                addiu   $t9, $s0, (sub_5E41AC - 0x5E0000)
                jalr    $t9 ; sub_5E41AC
                li      $a2, 0x27  # '''
                beqz    $v0, loc_4F58B0
                nop
                move    $s2, $v0
                b       loc_4F6208
                li      $a0, 1
```

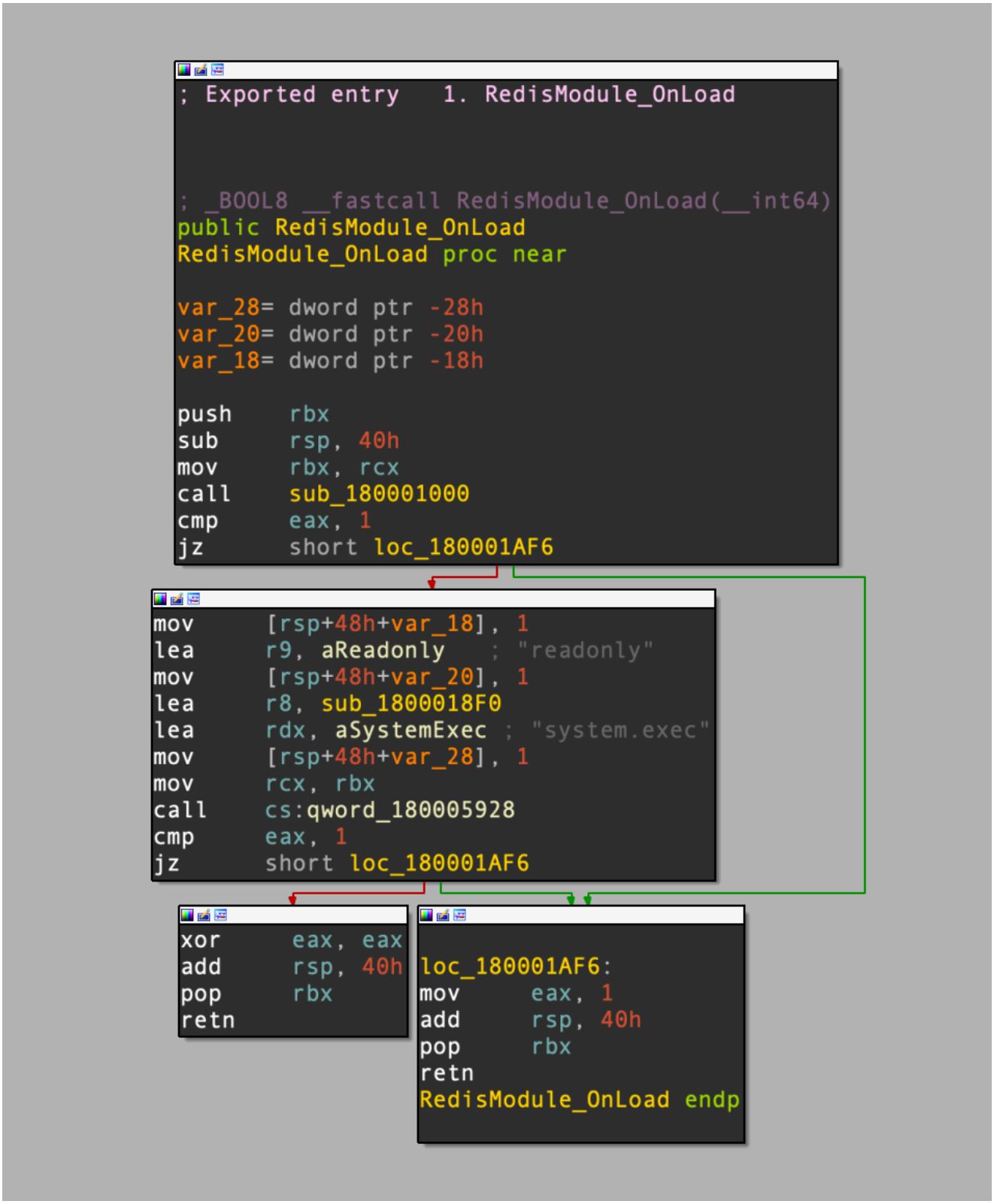
This procedure can be observed during dynamic analysis, with the binary utilising the `prctl()` syscall and passing the parameters `PR_SET_DUMPABLE`, `SUID_DUMP_DISABLE`.

```
munmap(0x77ff1000, 4096)                = 0
prctl(PR_SET_DUMPABLE, SUID_DUMP_DISABLE) = 0
prlimit64(0, RLIMIT_CORE, {rlim_cur=0, rlim_max=0}, NULL) = 0
```

Example strace output demonstrating disabling of core dumps

Embedded DLL

As mentioned in the Static Analysis section, the MIPS variant of P2Pinfect includes an embedded 64-bit Windows DLL. This DLL acts as a malicious loadable module for Redis, implementing the `system.exec` functionality to allow the running of shell commands on a compromised host.



Disassembly of the Redis module entrypoint, mapping the system.exec command to a handler

This is consistent with the previous examples of P2Pinfect, and demonstrates that the intention is to utilise MIPS devices for the Redis-specific initial access attack patterns mentioned throughout this blog.

Interestingly, this embedded DLL also includes a Virtual Machine evasion function, demonstrating the lengths that the P2Pinfect developers have taken to hinder the analysis process. In the DLLs main function, a call can be observed to a function helpfully labelled `anti_vm` by IDAs Lumina feature.

```
__int64 __fastcall dllmain_crt_dispatch(HINSTANCE a1, int a2, void *const a3)
{
  int v3; // edx
  __int64 v4; // rdx
  __int64 result; // rax

  if ( !a2 )
    return dllmain_crt_process_detach(a3 != 0i64);
  v3 = a2 - 1;
  if ( !v3 )
    return dllmain_crt_process_attach(a1, a3);
  v4 = (unsigned int)(v3 - 1);
  if ( (_DWORD)v4 )
  {
    if ( (_DWORD)v4 != 1 )
      return 1i64;
    LOBYTE(result) = _scrt_dllmain_crt_thread_detach(a1, v4, a3);
  }
  else
  {
    LOBYTE(result) = anti_vm();
  }
  return (unsigned __int8)result;
}
```

Decompiler output showing call to `anti_vm` function

Viewing the function itself, we can see researchers Christopher Gardner and Moritz Raabe have identified it as a known VM evasion method in other malware samples.

IDA's graph view for the `anti_vm` function showing Lumina annotations

Conclusion

P2Pinfect's continued evolution and broadened targeting are clearly the work of a determined and sophisticated threat actor. The cross-platform targeting and utilisation of a variety of evasion techniques demonstrate an above-average level of sophistication when it comes to malware development. Clearly, this is a botnet that will continue to grow until it's properly utilised by its operators.

While much of the functionality of the MIPS variant is consistent with the previous variants of this malware, the developer's efforts in making both the host and embedded executables as evasive as possible show a continued commitment to complicating the analysis procedure. The use of anti-forensics measures such as the disabling of core dumps on Linux systems also supports this.

Cado Security Labs researchers will continue to monitor and report on the growth of this emerging botnet.

If you'd like to see how Cado can help you investigate this threat,**request a demo**.

Indicators of Compromise (IoCs)

Files	SHA256
MIPS ELF	8b704d6334e59475a578d627ae4bcb9c1d6987635089790350c92eafc28f5a6c
Embedded DLL Redis Module	d75d2c560126080f138b9c78ac1038ff2e7147d156d1728541501bc801b6662f

About The Author

Matt Muir

Matt is a security researcher with a passion for UNIX and UNIX-like operating systems. He previously worked as a macOS malware analyst and his background includes experience in the areas of digital forensics, DevOps, and operational cyber security. Matt enjoys technical

writing and has published research including pieces on TOR browser forensics, an emerging cloud-focused botnet, and the exploitation of the Log4Shell vulnerability.

About Cado Security

Cado Security is the provider of the first cloud forensics and incident response platform. By leveraging the scale and speed of the cloud, the Cado platform automates forensic-level data capture and processing across cloud, container, and serverless environments. Only Cado empowers security teams to respond at cloud speed.

Prev Post Next Post