public-research/potassium/report.md at main · deepfield/public-research

By deepfield-ert

Archived: 2026-09-26 02:01:23 UTC

Potassium: it was never about the taste

Nokia Deepfield Emergency Response Team (ERT)

First published: 2026-05-20 | Last updated: 2026-05-21

> **Content warning:** This report quotes malware artifacts verbatim, including domain names, URL paths, campaign tags, and embedded strings chosen by the threat actor. Some contain crude or offensive language, including an antisemitic slur. These are reproduced exactly as found in samples to enable accurate detection and attribution.

Summary

Potassium is a Mirai-derived DDoS botnet, first publicly identified in March 2026 by Synthient and named after the C2 subdomain `potassium.vitacocoyougolocobecauseyouaresodamndeliciocobarampam[.]st`. Since then we have observed at least two additional campaign roots from the same operator: `ikhebkankerinmijnrechterteelbal[.]st` (the `iambig` variant, grammatically correct Dutch for "I have cancer in my right testicle", a phrase that uses *kanker* as Dutch internet slang for emphasis rather than its literal medical meaning), and most recently `botlesscucks[.]st` (the `botlesscucks` campaign, first observed 2026-05-18, which the operator brands internally with an antisemitic slur we use only as a verbatim IoC in Detection). Whether there are other campaign roots we have not seen is an open question; the operator clearly rotates infrastructure on a multi-week cadence. Three campaign roots, three crude jokes, one operator with strong opinions about what makes a good domain name.

What distinguishes Potassium from the dozens of Mirai forks we track is not its DDoS capability, which is standard, but its choices. The operator encrypts a 222-entry config table with ChaCha20, the first time we have seen ChaCha20 used for string-table obfuscation in a Mirai variant, while protecting C2 traffic with nothing more than a static XOR key (`0xED`). There is a reverse shell capability the operator branded `SHOUT` . There is a DNS anti-fingerprinting scheme that resolves to decoy IPs in unrelated ASNs and recovers the real C2 by swapping the two 16-bit halves of the address. And there is a banner string baked into the binary that reads: `it was never about the taste` . The result is a botnet that looks more thought-about than most Mirai forks and shipped anyway.

The original `vitacoco` C2 has not responded to our probes since late April. The `iambig` C2 (`45.153.34[.]245`) is reachable intermittently. The newest campaign, `botlesscucks` (`45.156.87[.]243` , Pfcloud / VMHeaven NL, staging at `117.55.203[.]189`), is the one currently dispatching attack commands to

connected bots, at a steady rate of roughly 300 commands per day. The first hour after the new C2 came online included FiveM lobbies on residential ISPs in Switzerland, Canada, Hungary, and the UK; an Arma 3 server on a Dutch host; and roughly seven Tor SOCKS ports on Google Cloud IPs.

Key findings

- **At least three campaigns from one codebase** (`vitacoco` , `iambig` , `botlesscucks`), each with a different C2 root domain and staging path but with byte-identical ChaCha20 key, ChaCha20 nonce, bot seed (`14861879`), XOR key (`0xED`), 19-byte registration header, killer-port bind (`127.0.0.1:1234`), and C2 protocol. The operator appears to roll the campaign root every three to four weeks.
- **ChaCha20 for config, XOR for the wire.** The choice to invest in real cryptography for the static string table while protecting in-flight C2 traffic with a single static byte is unusual. The threat model it implies (defeat static-string extraction; do not defeat network monitoring) is rare and idiosyncratic.
- **SHOUT reverse shell.** A named protocol (command `SHELL` , response `SHOUT`) that lets the operator run `/bin/sh -c <anything>` on infected devices and receive XOR-encoded stdout, capped at 4 KB. We have observed only DDoS commands in the wild, but the capability is present on every infected device.
- **DNS word-swap obfuscation.** Resolved A records point to decoys in unrelated ASNs (US DoD network space at Incirlik AFB, universities, government research networks, AWS, Verizon Business, Kazakh residential broadband). The bot swaps the two 16-bit halves of the resolved address before connecting. The actual C2 IP is never the one visible in DNS.
- **Active C2 monitored continuously since late April 2026**, observing attack commands, info-query heartbeats, and three campaign rotations.

Timeline

Date	Event
2026-02-12	`vitacocoyougolocobecauseyouaresodamndeliciocobarampam[.]st` registered (StanCo / Istanco)
2026-03-13	First sample observed (ReversingLabs)
2026-03-17	Public report by Synthient identifying the installer and C2 domain
2026-04-07	10-architecture rebuild batch uploaded to MalwareBazaar (x86, PPC, SH4, MIPS, ARM variants)
2026-04-22	`havanagila` staging build (`45.153.34[.]26`) shares the iambig C2 root
2026-04-24	`iambig` campaign variant appears from `92.38.186[.]44` ; C2 domain `ikhebkankerinmijnrechterteelbal[.]st`

Date	Event
2026-05-01	`iambig` rebuild (3.4 KB smaller), also served via netcat on port 25565; the rebuild switches to a new 11-port rotation pool
2026-05-09	Original `vitacoco` C2 not responding to TCP probes; iambig C2 active
2026-05-10	`havanagila` arm7 binary rebuilt (`cd8420b5...` , 144,388 bytes) at the same staging path `45.153.34[.]26/havanagila123/arm7`
2026-05-18	`botlesscucks` campaign first observed; C2 root `botlesscucks[.]st` , real C2 `45.156.87[.]243` , staging `117.55.203[.]189/jewishgoldowner/arm7`

Samples

Six representative hashes across multiple architectures (plus nine additional architecture variants from the April rebuild batch), all sharing byte-identical ChaCha20 key and nonce material:

Hash (prefix)	Architecture	Campaign	C2 domain
`6ef4ce02`	ARM	`vitacoco`	`potassium.vitacoco...st`
`cd29241e`	x86	`vitacoco`	same (10-arch rebuild)
`a87aa799`	ARM	`iambig`	`ikhebkankerinmijnrechterteelbal.st`
`3f13e18e`	ARM	`havanagila`	same (iambig sibling)
`6833cb46`	ARM	`iambig`	same (rebuild, 3.4 KB trimmed)
`cd8420b5`	ARM	`havanagila`	same (arm7 rebuild served from `45.153.34[.]26`)
`a75a61c1`	ARM	`botlesscucks`	`musika.botlesscucks.st` (+ `stoplooking1/2` fallbacks)

Across all three campaign variants the 32-byte ChaCha20 key (`5a3f2e7b...3c2b6a08`), 12-byte nonce (`4a1b00ff...00010305`), bot seed (`14861879`), XOR key (`0xED`), 19-byte registration header (`0a0b0c0d0e0f00030201020304050504030201`), killer-port bind (`127.0.0.1:1234`), and the post-2026-05-01 11-port rotation pool are byte-identical. Ghidra function-size comparison shows 94% overlap between `iambig` and `vitacoco` . These are rolling campaigns from the same operator on the same source tree.

All samples are statically linked ELF binaries compiled with Aboriginal Linux (GCC 4.2.1) and uClibc. Full hashes are in `iocs/hashes.csv` .

C2 architecture

DNS word-swap obfuscation

The DNS resolution layer includes an IP transformation that makes A records misleading to casual observers. When the bot resolves a C2 domain, the returned IP is word-swapped before connection: `A.B.C.D` becomes `C.D.A.B`. The actual C2 IP is never the one in DNS.

The `vitacoco` domain resolves to IPs such as `132.198.176.65`, which belongs to the University of Vermont (AS1351). The bot word-swaps this to `176.65.132.198`, the actual C2. Similarly, `132.246.176.65` (National Research Council of Canada, AS376) becomes `176.65.132.246`. An analyst checking DNS records without knowledge of the swap sees C2 traffic apparently bound for universities and government research institutions. The misdirection survives most automated enrichment pipelines. The whole obfuscation layer is one line of bitwise algebra, and the bar it has to clear is whatever enrichment runs by default.

The swap function from the binary (at `0x17ac8`):

```
uint swap_ip(uint ip) {
    return ((ip & 0xff0000) >> 8 | (ip >> 24) << 16) >> 8 |
           (ip & 0xff) << 16 |
           (((ip & 0xff00) << 8) >> 16) << 24;
}
```

This expression is algebraically equivalent to `(ip << 16) | (ip >> 16)`, a swap of the two 16-bit halves of the 32-bit word. The function has a single callsite, sitting between the in-binary DNS resolver and the `connect()` syscall. There is no other transformation between resolution and connection. Any sample whose `connect()` target IP is the halves-swap of one of its DNS-resolved IPs is almost certainly Potassium or a same-codebase variant.

The pattern survived every campaign rotation. The `botlesscucks` campaign uses it identically: `musika.botlesscucks[.]st` resolves to `87.243.45.156` (Kazakh residential broadband, AS21299), and the bot connects to `45.156.87.243` (Pfcloud / VMHeaven NL, AS51396).

Port rotation

Each connection attempt selects a random port from an 11-entry table embedded in the binary. The four samples analyzed in late April share an identical pool:

`19876, 60764, 27603, 27776, 29502, 1207, 26608, 7574, 876, 57536, 6428`

The `iambig` rebuild from 2026-05-01 onward and the entire `botlesscucks` campaign use a different pool, with all 11 ports listening simultaneously on the live C2:

`7193, 15987, 23789, 27651, 32876, 38429, 42061, 46852, 49376, 54123, 61543`

Combined with DNS round-robin (five-plus IPs) and the word-swap layer, a single connection attempt uses one of dozens of possible IP and port combinations. The two distinct pools across campaign generations mean an external prober tracking only the original pool will miss the newer infrastructure entirely.

Single-instance lock on TCP/1234

The bot binds a TCP listener on port 1234, bound to the egress IP discovered via the standard "UDP-connect to 8.8.8.8:53 + getsockname" trick, with 127.0.0.1 fallback. This is not a backdoor or an alternate C2 channel: the accept handler immediately closes the listener, kills its children, sleeps a second, and exits. A second bot instance launched on the same device hits `EADDRINUSE` on its own bind, connects to the existing listener to trigger that exit branch, and takes over. The wider pattern is Mirai's single-instance lock, but the dedicated port (vs. the more common `127.0.0.1:48101`) and the takeover semantics are characteristic enough to be useful as a host indicator.

Encryption

ChaCha20 config table (222 entries)

The static config table uses IETF ChaCha20 with a key and nonce embedded in the `.data` section:

Parameter	Value
Key (32 bytes)	`5a3f2e7b 8c9d0a6e 3b4f7c8a 1d5c2b0d 6f1a0c3e 2d8e4a09 1c7e5a4d 3c2b6a08`
Nonce (12 bytes)	`4a1b00ff 2d7c3a12 00010305`
Counter	1 (first 64-byte keystream block skipped)

This is the first Mirai variant we have observed using ChaCha20 for string-table obfuscation. Standard Mirai uses a 4-byte XOR key; most derivatives follow suit. The ChaCha20 choice makes static string extraction harder than a simple XOR brute-force, but the key is static in `.data`, identical across all builds, so offline decryption is straightforward once located. In key-rotation terms the operator's stance is one-and-done: the ChaCha20 key has aged better than the original C2 domain.

The 222 entries cover C2 domains, bot process names, an operator banner, process kill lists (100+ kernel threads and daemons), filesystem paths (50+ DVR and IoT-specific paths), architecture identifiers, shell paths, and reboot-blocking strings.

XOR C2 traffic (key 0xED)

All C2 wire traffic, including registration, attack commands, info-query responses, and SHOUT responses, uses a single static XOR byte. The contrast with the ChaCha20 config table is striking: the operator invested in real cryptography to protect strings at rest and protected strings in transit with the digital equivalent of ROT13. A network observer with `tcpdump` and `xxd` reads every command in real time, which the operator presumably knows and presumably does not care about.

What is different about the C2 protocol

The wire protocol is conventional Mirai underneath: TCP, select-based event loop, 15-second timeout, 2-byte plaintext keepalive every six timeouts, attack commands in the standard 4-byte duration plus 1-byte attack ID plus

N-target plus TLV-options binary format. We will not repeat that here. What is worth calling out is the handful of design choices that distinguish Potassium from other Mirai families we track.

Campaign tags in the registration packet. Every bot stamps itself with a short ASCII tag on registration (`wget.woof` , `curl.woof` , `xpl1` , `xpl2` , `xpl3` , `hoofdzak` , `iambig` , `jewishgold`). The `jewishgold` tag is an antisemitic slur chosen by the operator and reproduced verbatim only because it is a literal byte sequence on the wire; we use `botlesscucks` (the C2 root domain) as our analyst label for the same campaign throughout this report. The tag combines the download method that delivered the bot and the campaign name, giving the operator per-installer analytics on the C2 side. The combination of delivery vector and campaign in a single string is more granular than we typically see, though we have not gone looking for the equivalent in other families. Marketing-funnel attribution for botnet installs is the kind of thing the rest of the ecosystem may well be doing too, just less visibly.

Broken registration trailer the C2 accepts anyway. The 2-byte trailer at the end of every registration packet is computed by a routine at `0x134e0` in the binary — an Internet checksum (ones-complement sum of little-endian 16-bit words), not the CRC16 sometimes assumed. At the registration call site the routine is invoked with a halved byte count `(N + M + 4) >> 1` , so the checksum only protects roughly the front half of the XOR-encoded body. The live C2 treats the trailer permissively: a CRC16-Modbus over the full body is accepted, and so is the bot's exact broken halved-length output. The most natural reading is that C2-side validation of the trailer was loosened at some point after the bot code was originally compiled, taking the mismatch off the table without anybody having to ship a new bot build. The asymmetric posture — real ChaCha20 for the static config table, an unenforced 2-byte trailer on the wire — is consistent with the rest of the family's threat model.

Receive-side `RECV_MAGIC` for info queries. The C2 prefixes status-poll packets with a fixed 18-byte magic. The bot checks this prefix before falling through to the SHELL or attack-command parsers. The magic is data, not a header field, which makes it cheap to grep for in packet captures: searching `tcpdump` output for the XOR-decoded magic flushes every potassium info query on the wire.

SHOUT reverse shell on the same channel. Most Mirai forks ship a dispatcher and stop there. Potassium ships `SHELL` (C2 → bot, plaintext command up to 1010 bytes) and `SHOUT` (bot → C2, XOR-encoded stdout up to 4090 bytes) on the same TCP connection, framed by a 5-byte literal magic and a big-endian length. The implementation is plain `pipe()` + `fork()` + `execl("/bin/sh", "sh", "-c", cmd, NULL)` , with `/system/bin/sh` as an Android fallback. The capability changes the post-attack posture: an infected device is not just a DDoS source, it is a foothold the operator can use for credential collection, lateral movement, and persistence beyond the DDoS payload. The naming (`SHELL` in, `SHOUT` back) is the kind of onomatopoeia engineering you do when you have one shot at a memorable wire-format magic and you are not going to waste it.

Word-swap obfuscation, again. Covered above but worth repeating in this context: the bot does its own DNS resolution against `8.8.8.8:53` (bypassing the libc resolver entirely), then halves-swaps the result before `connect()` . Combined with five-IP DNS round-robins, this is the most effective obfuscation layer the family ships.

Attack methods

The attack module retains Mirai's standard binary command format (4-byte big-endian duration, 1-byte attack ID, 1-byte target count, N × 5-byte `<IP/prefix>` targets, 1-byte option count, TLV options). The dispatch table is 20 entries, each pointing to a flood handler whose actual socket type is verified from `socket()` calls in the disassembly. The dispatch IDs are reassigned relative to a reference Mirai source, so any tool that labels methods by stock-Mirai (ID → name) mapping will mislabel them. The protocol column below is verified from each handler's `socket()` arguments; the Name column shows the conventional Mirai name for that slot ID, which now points at a different handler in potassium. The Notes column flags name/protocol mismatches and handler quirks; empty cells indicate the name and the implementation agree. The dispatch table:

ID	Name	Protocol	Notes
0	`udp_plain`	UDP RAW	Crafted UDP flood
1	`udp_raw`	UDP DGRAM	DGRAM despite the name
2	`tcp_bypass`	TCP RAW	
3	`syn_spoof`	TCP RAW	
4	`ack_flood`	TCP STREAM	
5	`syn_data`	UDP RAW	RAW UDP despite the name
6	`tcp_socket`	UDP DGRAM	UDP despite the name
7	`socket_hold`	UDP DGRAM	UDP despite the name
8	`tcp_raw`	TCP RAW + STREAM	Two sockets per handler
9	`wra_flood`	ICMP RAW	ICMP, not TCP
10	`gre_ip`	GRE (IPPROTO_RAW)	GRE encapsulation
11	`tcp_stomp`	TCP RAW	
12	`ovh_bypass`	UDP DGRAM (connected)	`connect()`-locked, uses `send()`
13	`std_flood`	UDP DGRAM	Six-template payload rotator (see below)
14	`udp_bypass`	TCP RAW + STREAM	TCP despite the name
15	`udp_openvpn`	UDP DGRAM (connected)	`connect()`-locked
16	`dns_query`	TCP RAW	TCP despite the name
17	`mc_ping`	TCP RAW + STREAM	Minecraft-server ping flood
18	`tcp_xmas`	TCP RAW	TCP with all flags set
19	`http_connect`	TCP STREAM	L7 GET/POST with 3-UA rotation

Reading the table is a little like reading a Discord server channel list after a busy weekend of admin changes: the labels still mean what they originally meant; the channels do not.

The TLV option parser supports keys for fixed payload size (`0x00`), source and destination ports (`0x06` , `0x07`), TTL (`0x04`), connection count (`0x18`), packet-per-second cap (`0x22`), user-agent override (`0x24`), and HTTP method and path overrides (`0x14` , `0x15`). Options `0x1a` and `0x1b` are a min/max range for randomized per-packet payload size: the attack handler reads both, requires `0x1a ≤ 0x1b` , and picks a random length in `[0x1a, 0x1b]` for each packet it crafts. Typical observed values `0x1a=1312, 0x1b=1440` are a narrow window sitting just under common MTUs, which keeps the attack traffic from fragmenting. `0x00` (above) and `0x1a` / `0x1b` are alternative modes — fixed vs. randomized payload size — rather than redundant.

The `std_flood` handler (ID 13) rotates across six payload templates. Three of them mimic authenticated web traffic with `token=<40-hex>&guid=<24-hex>` strings. These are not real credentials; they are attack-payload content designed to look like legitimate web requests so DDoS-mitigation L7 fingerprinters spend time processing them.

Observed in the wild

From continuous monitoring of the active C2 since 2026-05-09, we have logged roughly **300 attack commands per day** alongside heartbeat info queries from the C2. Across the 10-day window 2026-05-09 to 2026-05-19 the dispatched methods break down as follows:

Attack ID	Name	Protocol	Share of dispatches
7	`socket_hold`	UDP	66.0%
4	`ack_flood`	TCP	17.5%
15	`udp_openvpn`	UDP	5.8%
14	`udp_bypass`	TCP	4.5%
12	`ovh_bypass`	UDP	2.6%
8	`tcp_raw`	TCP	1.2%
1	`udp_raw`	UDP	1.2%
13	`std_flood`	UDP	0.7%
19	`http_connect`	TCP	0.3%
9	`wra_flood`	ICMP	0.2%
3, 10	`syn_spoof` , `gre_ip`	TCP, GRE	<0.1% each

Eight handlers in the dispatch table (`udp_plain` , `tcp_bypass` , `syn_data` , `tcp_socket` , `tcp_stomp` , `dns_query` , `mc_ping` , `tcp_xmas`) were not dispatched during the window. They are present in the binary; the

operator did not invoke them in the observed period. If any of them ever fires we will be looking at the binary again, on Chekhov's gun rules.

Roughly a third of all attacks target gaming infrastructure (FiveM, Arma 3, Minecraft, Source-engine servers); the remainder split between general web (`80` / `443`), DNS, and miscellaneous services. Attack duration distribution: minimum 20 seconds, median 66 seconds, p95 180 seconds, single observed maximum of 3600 seconds. The 66-second median sits squarely in the range of the cheapest tier offered by most booter services we have priced; whether that reflects operator demos or paying customers picking the entry-level option, we cannot tell from the C2 traffic alone.

The gaming target list reads less like an infrastructure campaign and more like someone trying to ratio a specific Discord. FiveM, Arma 3, and Source-engine community servers are where individual disputes turn into paid outages.

The newest campaign (`botlesscucks`) has been live too briefly to draw firm conclusions, but its first hour of observed traffic skews to the same `socket_hold` / `ack_flood` / `udp_bypass` / `ovh_bypass` / `std_flood` mix as iambig, with target ports leaning slightly more toward gaming (`6672` repeated, plus new sightings on `9050` , `9011` , `9005` , `9026` , `9062` , `9116` , `9124` , which look like Tor SOCKS ports, a target class we had not previously seen this operator hit). GTA mod lobbies and Tor SOCKS endpoints share very little customer demographic, which makes the same-hour overlap one of the more curious patterns we have logged.

Bot behavior

Process management

- **Process disguise**: renames itself via `/proc/self/comm` to `dvrLocker` or `dvrInside` (from config table entries 4 and 5), blending in with DVR management processes.
- **Fake crash banner**: outputs `Segmentation fault (core dumped)` to stdout on startup (config table entry 9), followed by the actual banner `it was never about the taste` (entry 8). Anyone watching terminal output briefly believes the bot crashed; the motto two lines later somewhat undoes the bit.
- **Competitor killer**: scans `/proc/*/maps` for byte patterns from rival Mirai-family binaries and sends `SIGKILL` to any match. Standard Mirai-fork behaviour; the targets are the usual suspects.
- **Reboot prevention**: kills `reboot` , `shutdown` , `halt` , and `poweroff` processes.
- **Watchdog disable**: accesses `/dev/watchdog` and `/dev/misc/watchdog` to prevent hardware watchdog resets.
- **Daemonization**: double `fork()` with stdio redirect to `/dev/null` .

Scanner

Mirai-style telnet scanner with process-name awareness. The kill list contains 100+ kernel thread names, network daemons, and system processes that should not be terminated. ADB-based delivery to Android devices is observed in practice (the `execl("/system/bin/sh", ...)` fallback in the SHOUT handler is consistent with this), while the DVR/IoT-specific paths in the config table (`/dvr/bin` , `/mnt/mtd/app` , `/duksan` , `/userfs` , `/gm/bin` ,

`/var/Sofia`, `/home/davinci`) point at HiSilicon-based DVRs, NVRs, and IP cameras as an additional target device class.

Operator artifacts

The naming. The operator's naming conventions are a study in commitment to a bit. The parent domain (`vitacocoyougolocobecauseyouaresodamndeliciocobarampam[.]st`) is 60 characters of stream-of-consciousness brand enthusiasm for Vita Coco coconut water. The staging path `/mypantsarefullofshit/arm7` is exactly the kind of URL path that gets flagged in corporate proxy logs. The alternative staging path `/1000mgofpotassiumaday/arm7` is where the family gets its public name. The botnet is, in effect, named after a dietary supplement recommendation encoded in a URL path. There are worse precedents for botnet naming conventions, but not many.

The Dutch pivot. The `iambig` campaign is the only one of the three with Dutch content. Its C2 domain (`ikhebkankerinmijnrechterteelbal[.]st`) is native-level Dutch with correct grammar and word order, and the use of *kanker* as an intensifier (rather than its literal medical meaning) is characteristic of Dutch internet slang. The campaign tag `hoofdzak` ("main thing", observed 2026-05-09) is also Dutch. The other artifacts on the operator's side (`vitacocoyougoloco…`, `botlesscucks`, the motto, all staging paths, the `xpl*` / `wget.woof` / `curl.woof` tags) are in English. The honest reading is that the operator has at least Dutch fluency, not that they are primarily Dutch-speaking; one branded campaign in idiomatic Dutch is what the evidence supports.

The motto. Config table entry 8 reads `it was never about the taste`. Printed to stdout after the fake segfault. In context, it appears to be an operator motto, possibly referencing the Vita Coco theme (coconut water being famously divisive on taste). It also works as a statement of purpose for the botnet itself: the point was never the product, it was the infrastructure.

The bot seed. Every Potassium bot generates its unique ID by appending random digits to the seed `14861879`. The 8-digit number is consistent across all builds and campaigns. We have not identified a meaning.

Relationship to other families

Unrelated to the Aisuru / Jackskid ecosystem. No shared cryptography (no custom RC4, no XXTEA, no mbedTLS), no shared C2 protocol (raw TCP and XOR vs. their HTTP token/guid or TLS), no shared infrastructure, no shared developer artifacts.

Shared toolchain, not shared code. The Aboriginal Linux GCC 4.2.1 cross-compiler is used by Potassium, Flameblox, and Katana, but it is a publicly available Mirai build environment used by unrelated operators. The 10-architecture batch rebuild in April 2026 demonstrates the same pattern we see in Flameblox: an operator who found the Aboriginal multi-target build system and compiled for everything it supports.

Cross-family targeting overlap. Across our 10-day monitoring window, 124 potassium attack commands landed on a target IP within 60 seconds of an attack from a different family we track. Roughly half of those pairings (63 events) put potassium and CECbot on the same target within a second or two of each other, often on the same destination port. Lower-volume same-window overlap is also visible with Vibenet — a Mirai family we track internally but have not yet documented publicly — (46 events) and Drifter (14). We have not identified shared

infrastructure, shared code, or shared cryptographic material between potassium and any of these families. The cleanest reading is shared booter customers rather than shared operators: a single buyer paying multiple services to chain volume against the same target is the simplest explanation for simultaneous same-target same-port commands. This is the first time we have looked at cross-family target sync systematically; whether the pattern is specific to potassium or normal across the booter ecosystem is an open question we plan to widen. From the operator side this is two separate businesses each receiving a paid order; from the victim side it is one bad afternoon.

Detection

Network indicators

- TCP connections with XOR `0xED` encoded payloads.
- Registration packets containing the 19-byte fixed header `0a0b0c0d0e0f00030201020304050504030201`, a 1-byte plaintext tag length, an XOR-encoded body, and a 2-byte trailer (not strictly bound to a single checksum function — see C2 protocol).
- Connections to one of 11 fixed ports per variant. Pre-rebuild samples (all `vitacoco` builds plus original `iambig`): `876, 1207, 6428, 7574, 19876, 26608, 27603, 27776, 29502, 57536, 60764`. Post-rebuild `iambig` (2026-05-01 and later) and all `botlesscucks` traffic: `7193, 15987, 23789, 27651, 32876, 38429, 42061, 46852, 49376, 54123, 61543`.
- Local TCP listener on port 1234.
- SYN packets to port 23/TCP (the telnet scanner).
- Bot's own DNS resolution targeting `8.8.8.8:53` (the libc resolver is bypassed). The resolved IP is then halves-swapped before `connect()`.

Host indicators

- Process names `dvrLocker` or `dvrInside`.
- Banner strings `Segmentation fault (core dumped)` and `it was never about the taste`.
- Listener bound to port 1234 on the device's egress interface (or `127.0.0.1:1234` if the egress IP probe fails).

Indicators of compromise

Machine-readable IoC files are in `iocs/`:

File	Contents
`domains.csv`	C2 root and subdomain candidates per campaign
`ips.csv`	DNS-decoy IPs, word-swapped real C2 IPs, staging IPs
`hashes.csv`	Representative sample SHA-256 hashes per campaign and architecture
`keys.csv`	ChaCha20 key/nonce/counter, XOR key, bot seed, killer port

Prior research and acknowledgements

The Potassium family was first publicly identified by **@deobfuscately** (Synthient) on 2026-03-17, who named it after the staging URL path `/1000mgofpotassiumaday/arm7` and posted the original sample SHA-256 (`6ef4ce02...`). Without that initial pivot we would not have had a starting point.

We previously posted a public update on 2026-05-07 confirming the `iambig` variant, three sample hashes, the byte-swap C2 mechanism, and the Dutch C2 domain. One framing in that update has since been corrected by the verification work in this report: the C2 protocol is raw TCP with `0xED` XOR, not HTTP — the `token=` / `guid=` strings are attack-payload content (see Attack methods), not C2 metadata. ADB-based delivery to Android devices, also mentioned in that post, remains consistent with our observations; the encrypted config table additionally references HiSilicon DVR/NVR/IP camera filesystem paths, so the target device class is broader than Android TV alone.

The reverse engineering, protocol verification, encryption analysis, attack-table verification, three-campaign attribution, DNS word-swap mechanism, live monitoring infrastructure, and the cross-family same-target overlap measurement in this report are original Nokia Deepfield ERT analysis. Errors in this report are ours; corrections welcome.

Edit history

Date	Change
2026-05-21	C2 protocol: documented registration trailer (broken halved-length Internet checksum; live C2 accepts other forms, indicating server-side validation was loosened post-compilation); corrected TLV options `0x1a` / `0x1b` from speculative "timing/rate hints" to min/max range for randomized payload size.
