# fast16 | Mystery ShadowBrokers Reference Reveals High-Precision Software Sabotage 5 Years Before Stuxnet

By Vitaly Kamluk & Juan Andrés Guerrero-Saade

Published: 2026-04-23 · Archived: 2026-05-07 02:11:37 UTC

## Executive Summary

- SentinelLABS has uncovered a previously undocumented cyber sabotage framework whose core components date back to 2005, tracked as fast16.
- `fast16.sys` selectively targets high-precision calculation software, patching code in memory to tamper with results. By combining this payload with self-propagation mechanisms, the attackers aim to produce equivalent inaccurate calculations across an entire facility.
- This 2005 attack is a harbinger for sabotage operations targeting ultra expensive high-precision computing workloads of national importance like advanced physics, cryptographic, and nuclear research workloads.
- fast16 predates Stuxnet by at least five years, and stands as the first operation of its kind. The use of an embedded customized Lua virtual machine predates the earliest Flame samples by three years.
- The name ‘fast16’ is referenced in the infamous ShadowBrokers’ leak of NSA’s ‘Territorial Dispute’ components. An evasion signature instructs operators: “fast16 *** Nothing to see here – carry on ***”

## Overview

Our investigation into fast16 starts with an architectural hunch. A certain tier of apex threat actors has consistently relied on embedded scripting engines as a means of modularity. Flame, Animal Farm’s Bunny, ‘PlexingEagle’, Flame 2.0, and Project Sauron each built platforms around the extensibility and modularity of an embedded Lua VM. We wanted to determine whether that development style arose from a shared source, so we set out to trace the earliest sophisticated use of an embedded Lua engine in Windows malware.

Lua is a lightweight scripting language with a native proficiency for extending C/C++ functionality. Given the appeal of C++ for reliable high-end malware frameworks, this capability is indispensable to avoid having to recompile entire implant components to add functionality to already infected machines. We did not find an indication of direct shared provenance, but our investigation did uncover the oldest instance of this modern attack architecture.

Lua leaves a distinctive fingerprint. Compiled bytecode containers start with the magic bytes `1B 4C 75 61` ( `\x1bLua` ), followed by a version byte, and the engine typically exposes a characteristic C API and environment variables such as `LUA_PATH` . Hunting for these traits across mid-2000s malware collections surfaced a sample that initially looked unremarkable: `svcmgmt.exe` .

## svcmgmt.exe | A 2005 Lua-Powered Service Binary

On the surface, `svcmgmt.exe` appears to be a generic console-mode service wrapper from the Windows 2000/XP era.

| | |
|---|---|
| Filename | svcmgmt.exe |
| Filesize | 315,392 bytes |
| MD5 | dbe51eabebf9d4ef9581ef99844a2944 |
| SHA1 | de584703c78a60a56028f9834086facd1401b355 |
| SHA256 | 9a10e1faa86a5d39417cae44da5adf38824dfb9a16432e34df766aa1dc9e3525 |
| Type | PE32 executable for MS Windows 4.00 (console), Intel i386 |
| Link Time | 2005-08-30 18:15:06 UTC |

A closer look reveals an embedded Lua 5.0 virtual machine and an encrypted bytecode container unpacked by the service entry point.

The developers extended the Lua environment to include:

- a `wstring` module for native unicode handling
- a built-in symmetric cipher, exposed through a function commonly labelled `b`, used to decrypt embedded data
- multiple modules that bind directly into Windows NT filesystem, registry, service control, and network APIs.

Even by itself, `svcmgmt.exe` already looks like an early high-end implant, a modular service binary that hands most of its logic to encrypted Lua bytecode. The binary includes a crucial detail: a PDB path that links the binary to the kernel driver `fast16.sys`.

## fast16 | A Nagging Mystery from the ShadowBrokers Leak

Buried in the binary's strings is a PDB reference:

```
C:\buildy\driver\fd\i386\fast16.pdb
```

At first glance, the path is structured like any other compiler artifact: an internal build directory, a component name (fast16), and an architecture hint (i386). However, in this case there's a mismatch. The string appears inside of a service-mode executable, and yet the `driver\fd\i386\fast16` segment of the pdb string clearly refers to a kernel driver project.

Following that clue led us to a second binary, `fast16.sys`:

| | |
|---|---|
| Filename | fast16.sys |

| | |
|---|---|
| Filesize | 44,580 bytes |
| MD5 | 0ff6abe0252d4f37a196a1231fae5f26 |
| SHA256 | 07c69fc33271cf5a2ce03ac1fed7a3b16357aec093c5bf9ef61fbfa4348d0529 |
| Type | PE32 executable for MS Windows 5.00 (native), Intel i386, 5 sections |
| Link Time | 2005-07-19 15:15:41 UTC (0x42dd191d) |

This kernel driver is a boot-start filesystem component that intercepts and modifies executable code as it's read from disk. Although a driver of this age will not run on Windows 7 or later, for its time `fast16.sys` was a cut above commodity rootkits thanks to its position in the storage stack, control over filesystem I/O, and rule-based code patching functionality.

In April 2017, almost 12 years after the compilation timestamp, the same filename, "fast16" appeared in the ShadowBrokers leak. Dr. Boldizsár Bencsáth's research into Territorial Dispute points to a text file, `drv_list.txt`. The 250KB file is a short list of driver names used to mark potential implants cyber operators might encounter on a target box as "friendly" or to "pull back" in order to avoid clashes with competing nation-state hacking operations.

```
"*** MISTYVEAL ***","nethdlr"
"*** NETSPYDER ***","khlp807w"
"*** NOTHING TO SEE HERE - CARRY ON ***","fast16"
"*** OLYMPUS ***","Fdisk"
"*** PEDDLECHEAP ***","appinit"
```

Screenshot from Crysys Lab's ShadowBrokers leak analysis paper

The guidance for one particular driver, 'fast16', stands out as both unique and particularly unusual.

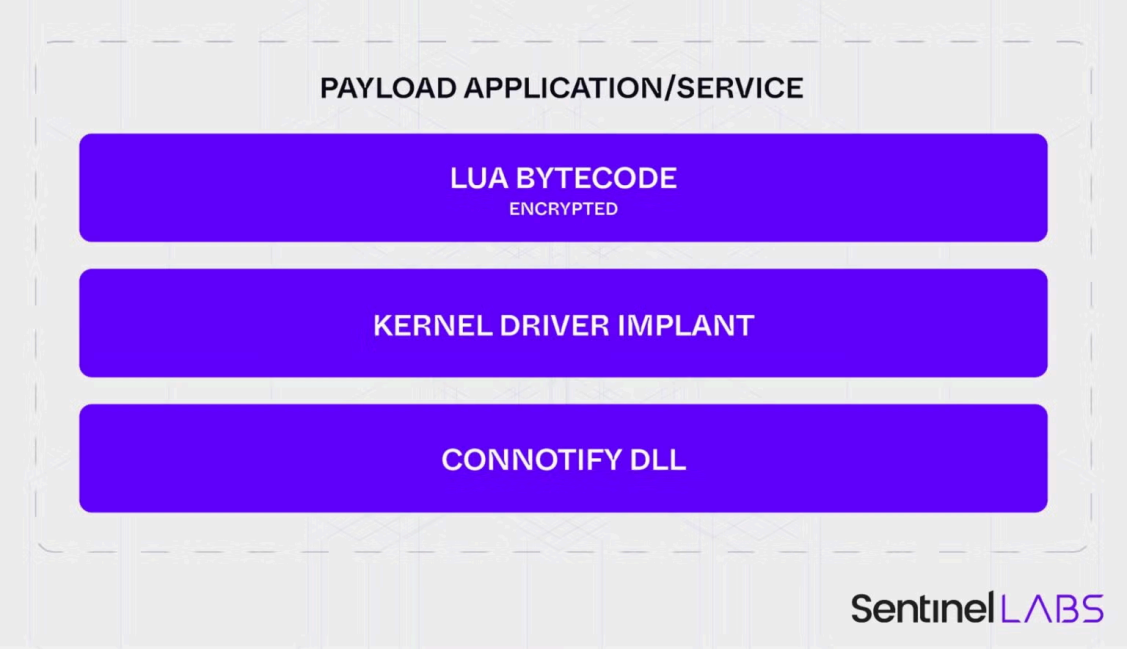
The string inside `svcmgmt.exe` provided the key forensic link in this investigation. The pdb path connects the 2017 leak of deconfliction signatures used by NSA operators with a multi-modal Lua-powered 'carrier' module compiled in 2005, and ultimately its stealthy payload: a kernel driver designed for precision sabotage.

## svcmgmt.exe | Architecture of the Carrier

The core component of fast16, `svcmgmt.exe`, functions as a highly adaptable carrier module, changing its operational mode based on command-line arguments.

- No arguments: Runs as a Windows service.
- `-p` : Sets InstallFlag = 1 and runs as a service (Propagate/Install & Run).
- `-i` : Sets InstallFlag = 1 and executes Lua code (Install & Execute Lua).
- `-r` : Executes Lua code without setting the install flag (Execute Lua).
- Any other argument ( `<filename>` ): Interprets as a filename, and spawns two children: the original command and one with the `-r` argument (Wrapper/Proxy Mode).

Internally, `svcmgmt.exe` stores three distinct payloads, including encrypted Lua bytecode that handles configuration, its propagation and coordination logic, auxiliary `ConnotifyDLL`, and the `fast16.sys` kernel driver.




Composition of the Carrier payload

By separating a relatively stable execution wrapper from encrypted, task-specific payloads, the developers created a reusable, compartmentalized framework that they could adapt to different target environments and operational objectives while leaving the outer carrier binary largely unchanged across campaigns.

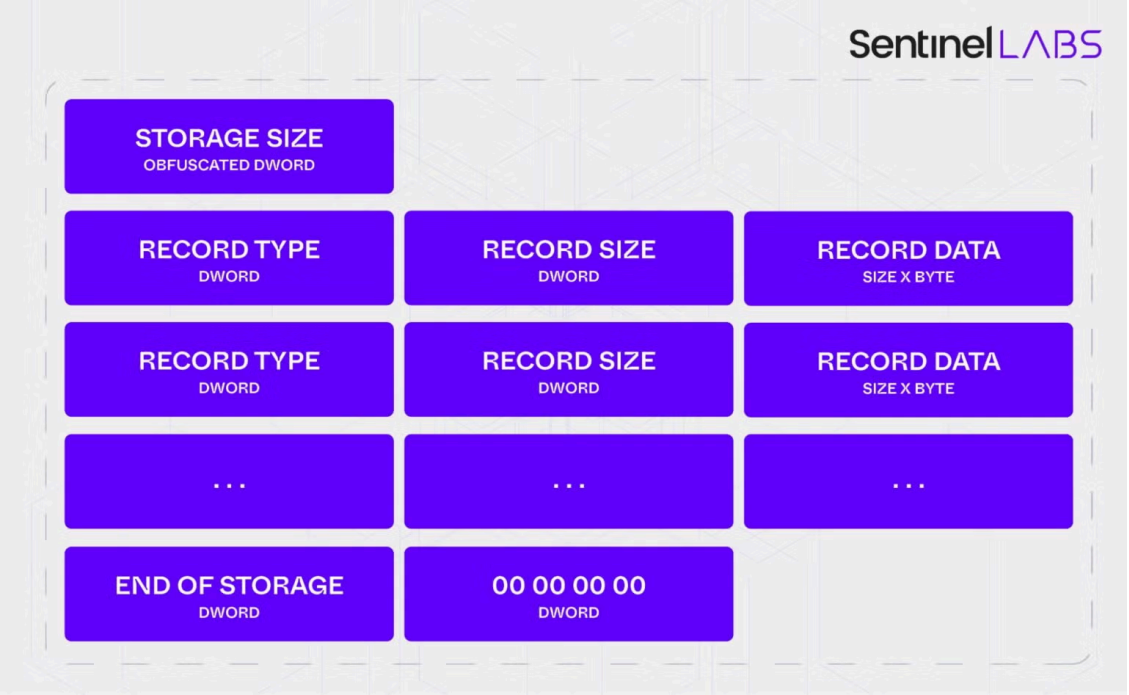
## The Wormlets and Early Evasion Architecture

The early 2000s saw a large number of network worms. Most were written by enthusiasts, spread quickly, and carried little or no meaningful payload. fast16 originates from the same period but follows a completely different pattern indicative of its provenance as state-level tooling. It's the first recorded Lua-based network worm, and was built with a highly specific mission.

The carrier was designed to act like cluster munition in software form, able to carry multiple wormable payloads, referred to internally as 'wormlets'. The `svcmgmt.exe` module performs the following steps:

1. Prepares the configuration, defining the payload path, service details, and target IP ranges.
2. Converts the configuration values to wide-character strings for the C layer.
3. Escalates privileges and installs the carrier executable as the `SvcMgmt` service, then starts it.
4. Optionally, based on the configuration setting, deploy the kernel driver implant `fast16.sys`.
5. Releases the wormlets. In this particular configuration, only one wormlet slot is populated with an SCM wormlet that looks for network servers, copies the payload over a network share and starts that remote service.
6. Repeats the process indefinitely, sleeping for the configured initial delay between waves, until a failure threshold or external kill condition is reached.

The wormlets were stored in the carrier's internal storage:




Structure of the internal storage

The single deployed wormlet found in `svcmgmt.exe` (the SCM wormlet) exemplifies a simple but effective propagation strategy based on native Windows capabilities and weak network security. It targets Windows 2000/XP environments and relies on default or weak administrative passwords on file shares. All spreading is done through standard Windows service-control and file-sharing APIs, an early example of propagation that leans on built-in administration features rather than custom network protocols.

Before this workflow runs, a pre-installation kill-switch checks the environment. The `ok_to_install()` routine calls `ok_to_propagate()` and propagation is only allowed if it's manually forced or if it's made sure common security products aren't found by checking for associated registry keys. The routine walks a list of vendor keys and aborts installation if any of them are present, preventing deployment into monitored environments.

For tooling of this age, that level of environmental awareness is notable. While the list of products may not seem comprehensive, it likely reflects the products the operators expected to be present in their target networks whose detection technology would threaten the stealthiness of a covert operation:

```
HKLM\SOFTWARE\Symantec\InstalledApps
HKLM\SOFTWARE\Sygate Technologies, Inc.\Sygate Personal Firewall
HKLM\SOFTWARE\TrendMicro\PFW
HKLM\SOFTWARE\Zone Labs\TrueVector
HKLM\SOFTWARE\F-Secure
HKLM\SOFTWARE\Network Ice\BlackIce
HKLM\SOFTWARE\McAfee.com\Personal Firewall
HKLM\SOFTWARE\ComputerAssociates\eTrust EZ Armor
HKLM\SOFTWARE\RedCannon\Fireball
HKLM\SOFTWARE\Kerio\Personal Firewall 4
```

```
HKLM\SOFTWARE\KasperskyLab\InstalledProducts\Kaspersky Anti-Hacker
HKLM\SOFTWARE\Tiny Software\Tiny Firewall
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Look n Stop 2.05p2
HKCU\SOFTWARE\Soft4Ever
HKLM\SOFTWARE\Norman Data Defense Systems
HKLM\SOFTWARE\Agnitum\Outpost Firewall
HKLM\SOFTWARE\Panda Software\Firewall
HKLM\SOFTWARE\InfoTeCS\TermiNET
```

A separate user-mode component, `svcmgmt.dll`, provides a minimal reporting channel. Contained within the carrier's internal storage, this DLL is registered through the Windows `AddConnectNotify()` API so that it's called each time the system establishes a new network connection using the Remote Access Service (RAS), responsible for dial-up connections and early VPNs in the 2000s.

| | |
|---|---|
| Module Name | User Module (connotifydll) |
| Filename | svcmgmt.dll |
| Filesize | 45056 bytes |
| MD5 | 410eddfc19de44249897986ecc8ac449 |
| SHA256 | 8fcb4d3d4df61719ee3da98241393779290e0efcd88a49e363e2a2dfbc04dae9 |
| Link Time | 2005-06-06 18:42:45 UTC |
| Type | PE32 DLL (i386, 4 sections) |

When invoked, the DLL decodes an obfuscated string to obtain the named pipe `\\.\pipe\p577`, attempts to connect to the local pipe, and writes the remote and local connection names to the pipe before closing it. The module doesn't run independently and must be registered by a host process.

## fast16.sys | A Filesystem Driver for Precision Sabotage

The kernel driver `fast16.sys` is the most potent component of the framework.

The driver is configured with `Start=0` (boot) and `Type=2` (filesystem driver) in the `SCSI` class group. It loads automatically at an early stage, alongside disk device drivers, and inserts itself above each filesystem device (NTFS, FAT, MRxSMB). On entry it:

- disables the Windows Prefetcher by setting the `EnablePrefetcher` value to **0** under the Session Manager's `PrefetchParameters` key, forcing subsequent code-page requests through the full filesystem stack,
- resolves kernel APIs dynamically using a simple XOR-based string cipher and a scan of `ntoskrnl.exe`, and
- exposes `\Device\fast16` and `\??\fast16` with a custom `DeviceType` value **0xA57C**, which serves as a secondary forensic marker.

The driver registers with `IoRegisterFsRegistrationChange` so it can attach a worker device object on top of every active and newly created filesystem device. All relevant I/O Request Packets, including `IRP_MJ_CREATE`, `IRP_MJ_READ`, `IRP_MJ_CLOSE`, `IRP_MJ_QUERY_INFORMATION`, `IRP_MJ_FILE_SYSTEM_CONTROL`, and associated Fast I/O paths, are routed through these worker devices.

Despite loading at boot, the kernel-level code injection engine is only activated after the system opens `explorer.exe`. This design defers expensive monitoring and patching until the desktop environment is available and avoids unnecessary impact on core boot performance.

## Narrow Targeting via Intel Compiler Artefacts

Once activated, `fast16.sys` focuses on executable files. A file is a valid target if it meets two criteria:

1. The filename ends with `.EXE`.
2. Immediately after the last PE section header, there is a printable ASCII string starting with `Intel`.

This selection logic points to executables compiled with the Intel C/C++ compiler, which often placed compiler metadata in that region. It indicates that the developers knew their target software was built with this toolchain.

For files meeting these criteria, the driver performs a PE header modification in memory. It injects two additional sections, `.xdata` and `.pdata`, and fills them with bytes from the original code section, increasing the section count and keeping a clean copy of the code. The intent is likely to increase stability while still allowing extensive patching, although without identifying the original target binaries this remains an informed hypothesis.

## Rule-Driven Patching and Floating-Point Corruption

The patching engine is a minimalist, performance-optimised, stateful scanning and modification tool. It is configured with a set of 101 rules, each containing pattern matching and replacement logic. To maintain performance, the engine:

- uses a 256-byte dispatch array and only flags the starting byte values of a small number of unique patterns,
- allows wildcards inside patterns so a single rule can match several compiler-optimised variants of the same code, and
- supports state flags that some rules can set or check, enabling multi-stage modification sequences similar to those used by advanced antivirus scanning engines.

Most patched patterns correspond to standard x86 code used for hijacking or influencing execution flow. One injected block is different. It’s a larger and complex sequence of Floating Point Unit instructions dedicated to precision arithmetic and scaling values in internal arrays. This code is a standalone mathematical calculation function unrelated to code flow hijacking or any other typical malicious code injection.

To understand what the driver expected to see, we converted the patching rules into hexadecimal YARA signatures and ran them against a large, period-appropriate corpus. The results showed a very low hit rate: fewer than ten files matched two or more patterns. Those matches, however, shared a clear theme. They were precision calculation tools in specialised domains such as civil engineering, physics and physical process simulations.

The FPU patch in `fast16.sys` was written to corrupt these routines in a controlled way, producing alternative outputs. This moves fast16 out of the realm of generic espionage tooling and into the category of strategic sabotage. By introducing small but systematic errors into physical-world calculations, the framework could undermine or slow scientific research programs, degrade engineered systems over time or even contribute to catastrophic damage.

A sabotage operation of this kind would be foiled by verifying calculations on a separate system. In an environment where multiple systems shared the same network and security posture, the wormable carrier would deploy the malicious driver module to those systems as well, reducing the chance that an independent calculation would diverge from the corrupted output.

At this time, we've been unable to identify all of the target binaries in order to understand the nature of the intended sabotage. We welcome the contributions of the larger infosec research community and have included YARA rules to hunt for these patterns in the appendix below.

## The Data Patching Engine

Even after deep analysis, fast16's driver looks deceptively simple. Beneath that minimal code is a rule-driven in-memory engine that quietly patches executable code as files are read from disk.

The engine relies on a compact set of just over a hundred pattern-matching rules and a small dispatch table so it only inspects bytes that are likely to matter. Most patterns correspond to ordinary x86 instructions, but one stands out: a larger block of floating-point (FPU) code dedicated to precision arithmetic. This injected routine scales values in three internal arrays passed into the function, subtly changing calculations.

```
.data:00014A5E DD 04 83                  fld     qword ptr [ebx+eax*4]
.data:00014A61 DD 99 D4 0F 00 00         fstp    qword ptr [ecx+0FD4h]
.data:00014A67 DC A1 D4 0F 00 00         fsub    qword ptr [ecx+0FD4h]
.data:00014A6D DD 81 A8 0F 00 00         fld     qword ptr [ecx+0FA8h]
.data:00014A73 DC 89 9C 0F 00 00         fmul    qword ptr [ecx+0F9Ch]
.data:00014A79 DD 91 B0 0F 00 00         fst     qword ptr [ecx+0FB0h]
.data:00014A7F BB D0 0E 00 00            mov     ebx, 0ED0h
.data:00014A84 01 CB                     add     ebx, ecx
.data:00014A86 DD 04 83                  fld     qword ptr [ebx+eax*4]
.data:00014A89 DC B1 58 0F 00 00         fdiv    qword ptr [ecx+0F58h]
.data:00014A8F DC 89 9C 0F 00 00         fmul    qword ptr [ecx+0F9Ch]
.data:00014A95 DD 99 B8 0F 00 00         fstp    qword ptr [ecx+0FB8h]
.data:00014A9B DC A1 B8 0F 00 00         fsub    qword ptr [ecx+0FB8h]
.data:00014AA1 DE F9                     fdivp   st(1), st
.data:00014AA3 DD 91 C0 0F 00 00         fst     qword ptr [ecx+0FC0h]
.data:00014AA9 DC 89 B0 0F 00 00         fmul    qword ptr [ecx+0FB0h]
.data:00014AAF D9 E0                     fchs
.data:00014AB1 DE C1                     faddp   st(1), st
.data:00014AB3 DD 99 C8 0F 00 00         fstp    qword ptr [ecx+0FC8h]
.data:00014AB9 5B                        pop     ebx
```

Injected FPU-based calculations

Without knowing the exact binaries and workloads being patched, we can't fully resolve what those arrays represent, only that the goal is to tamper with numerical results, not unauthorized access, malware propagation or other common malware objectives.

## The Patch Targets

Our best clues about the intended victims come from matching these patterns against large, era-appropriate software corpora. The strongest overlaps point to three high-precision engineering and simulation suites from the mid-2000s: LS-DYNA 970, PKPM, and the MOHID hydrodynamic modeling platform, all used for scenarios like crash testing, structural analysis, and environmental modeling.

LS-DYNA in particular has been cited in public reporting on Iran's suspected violations of Section T of the JCPOA, in studies of computer modeling relevant to nuclear weapons development.

> For example, a 2018 study by Kasaie used the LS-DYNA code to "compare the influence of Octol and PBXN-110 on the performance of a specific shaped charge warhead…PBXN-110 exhibited a weak penetration effect due to its lower density and detonation velocity compared to Octol." Studies with Octol are of interest, as it was a main explosive used in and mass produced for Iran's AMAD program.[71]
>
> 
>
> **Figure 6.** A high explosive casting mold from the Nuclear Archives.

Use of LS-DYNA code to research explosive payloads for Iran's AMAD program

## Compiler Footprints and Lineage

As we sought to understand the lineage of this unusual set of components, we noticed a quirk. Strings of the form `@(#)par.h $Revision: 1.3 $` inside the binaries point to an unusual source-control convention. The `@(#)` prefix is characteristic of early Unix Source Code Control System (SCCS) or Revision Control System (RCS) tooling from the 1970s and 1980s. These markers do not affect execution and are redundant in modern Windows kernel drivers.

Finding SCCS/RCS artefacts in mid-2000s Windows code is rare. It strongly suggests that the authors of this framework were not typical Windows-only developers. Instead, they appear to have been long-term engineers whose culture and toolchain came from older, high-security Unix environments, often associated with government or military-grade work. This detail supports the view that fast16 came from a well-resourced, long-running development program.

## A Digital Fossil with Modern Implications

`svcmgmt.exe` was uploaded to VirusTotal nearly a decade ago. It still receives almost no detections: one engine classifies it as generally malicious, and even that with limited confidence. For a stealthy self-propagating carrier that deploys one of the most sophisticated sabotage drivers of its era, that detection record is notable.

Together with its appearance in the ShadowBrokers 'Territorial Dispute' (TeDi) signatures, fast16 forces a re-evaluation of our historical understanding of the timeline of development for serious covert cyber sabotage operations. The code shows that:

- state-grade cybersabotage against physical targets was fully developed and deployed by the mid-2000s,
- embedded scripting engines, narrow compiler-based targeting and kernel-level patching formed a coherent architecture well ahead of better-known families, and
- some of the most important offensive capabilities in the ecosystem may still sit in collections as 'old but interesting' samples lacking the context to highlight their true significance.

Internally, the operation leaves very little in the way of branding. One of the few human-readable labels is wry and understated:

```
*** Nothing to see here - carry on ***
```

For many years there were no public write-ups, no named campaign and no headline incident linked to this framework.

In the broader picture of APT evolution, fast16 bridges the gap between early, largely invisible development programs and later, more widely documented Lua- and LuaJIT-based toolkits. It is a reference point for understanding how advanced actors think about long-term implants, sabotage, and a state's ability to reshape the physical world through software. fast16 was the silent harbinger of a new form of statecraft, successful in its covertness until today.

## Acknowledgements

SentinelLABS would like to thank Silas Cutler and Costin Raiu for their contributions along the way. We dedicate this research to the memory of Sergey Mineev, APT hunter extraordinaire, who pioneered many of the techniques that enabled this discovery.

## Appendix: Patching Engine Patterns and Target Candidates

### Extracted Match Patterns

```
48 89 84 24 9C 00 00 00 4B 0F 8F 79 FF FF FF 00
D8 E1 D9 5D FC D9 04 00
55 8B EC 83 EC 14 53 56 57 8B 3D ?? ?? ?? ?? 8B 0D 00
89 4D C8 8B FB 8B C8 00
```

```
8B 4C 24 0C 8B 01 83 F8 63 00
39 2D ?? ?? ?? ?? 0F 84 F4 00 00 00 8B 35 ?? ?? ?? ?? 2B 35
7C 02 89 C6 89 35 ?? ?? ?? ?? 89 B4 24 D0
83 3D ?? ?? ?? ?? 00 0F 84 70 BD FF FF 00
BE 07 00 00 00 BF 04 00 00 00 BB 02 00 00 00 00
8B 4D 10 C1 E2 04 8B 19 83 EA 30 8B CB 49
8D 1D ?? ?? ?? ?? 52 8D 05 ?? ?? ?? ?? 51 8D 15 ?? ?? ?? ?? 8D 0D ?? ?? ?? ?? 53 50 52 51 56 57 E8 ?
0F 8F A5 00 00 00 A1 ?? ?? ?? ?? 83 F8 14 7D 0D
8B 5D B0 0F 85 ?? ?? ?? ?? 8D 34 9D ?? ?? ?? ?? 8D 14 9D 00 0F 8E 1B 03 00 00 D9 05
8B 45 44 6B 00 04 D9 05 ?? ?? ?? ?? D8 B0
E9 7E 04 00 00 8B 74 24 1C 8B 54 24 14 85
83 39 63 0F 85 21 03 00 00 8B EE 85 F6 0F
85 DB 8B 55 D4 75 2C 89 35 00
75 18 8D 35 ?? ?? ?? ?? 56 8D 3D 00
8D 1D ?? ?? ?? ?? 52 8D 05 ?? ?? ?? ?? 51 8D 15 ?? ?? ?? ?? 8D 0D ?? ?? ?? ?? 53 50 52 51 56 57 E8 ?
D8 34 85 ?? ?? ?? ?? 8B 44 ?? ?? 8B CA 00
8B 5D 0C 8B 55 08 8B 36 8B 00
8D 04 BD ?? ?? ?? ?? 03 DF 00
8B EE 85 F6 0F 8E ?? ?? ?? ?? 8D 1C BD 00
D9 04 9D ?? ?? ?? ?? 83 ED 04 05 10 00 00 00 D8 0D 00
75 2C 89 35 ?? ?? ?? ?? 89 05 ?? ?? ?? ?? 89 15
89 55 F4 8B F9 8B D3 03 FB C1 E2 02 89 35
40 23 72 65 63 24 65 69 69 6E 20 2E 30 24 D9 5D 00 D9 03 D8 0D ?? ?? ?? ?? D8 0D 00
DF E0 F6 C4 41 A1 ?? ?? ?? ?? 74 5A
FF 35 ?? ?? ?? ?? E8 ?? ?? ?? ?? 9D D9 E0 D9 1D ?? ?? ?? ?? 8B 4C
6A 46 68 ?? ?? ?? ?? E8 ?? ?? ?? ?? 6A 03
D8 05 ?? ?? ?? ?? D9 55 00 9C
C2 08 00 A1 ?? ?? ?? ?? 8B 0C 85 ?? ?? ?? ?? 89 0E 00
83 EC 04 53 E8 ?? ?? ?? ?? EB 09 83 EC 04 53 00
D8 1D ?? ?? ?? ?? DF E0 F6 C4 41 B8 00 00 00 00 75 05 B8 01 00 00 00 85 C0 74 11 6A 29 00
2B DA 89 3C 03 83 3D 00
D9 5D C0 8B 4D C0 D9 45 E0 89 0E 00
8B 05 ?? ?? ?? ?? 8B 0D ?? ?? ?? ?? 0F 85 7E 00 00 00 0F AF 15 00
B9 01 00 00 00 C1 E7 02 8B BF ?? ?? ?? ?? 8B D7 85 FF 8B 55 30 8B 45 30 D8 C9 8B 75 2C 00 9A 8B 00 0
2B FB 8B DE C1 E3 02 89 7D A0 03 5D A0 8B 03 F7 F7 DB 0C 02 89 35
0F 0F 94 C0 23 C3 33 D2
8B 55 30 8B 75 2C D8 C9 8B 45 30 00
DD 05 ?? ?? ?? ?? 8B 05 ?? ?? ?? ?? 8B 15 ?? ?? ?? ?? 0F AF 05 ?? ?? ?? ?? 8B 1D ?? ?? ?? ?? 0F AF 1
68 28 00 00 00 57 E8 ?? ?? ?? ?? 8B 1D ?? ?? ?? ?? 8B 35 ?? ?? ?? ?? 0F AF 1D ?? ?? ?? ?? 8B 3D ?? ?
8B 75 38 8B 4D 34 D8 C9 8B 00
8B 55 88 8B 5D B0 83 7D 84 01
55 8B EC 83 EC 2C 33 D2 53 56 57 8B
55 8B EC 83 EC 2C B9 46 00 00 00 53 56 57 8B 00
```

## Patch Target Candidate 1: LS-DYNA 970 Software Suite

The LS-DYNA suite is powerful engineering simulation software used to analyze how materials and structures behave under extreme conditions. The tool is used by engineers to simulate physical events and model conditions while avoiding expensive or dangerous experiments.

LS-DYNA is designed for handling dynamic, complex events that occur at speed, such as car crashes, explosions, impacts, metal forming, and manufacturing processes. It was commonly used by automotive companies, aerospace engineering, defense and military research, as well as manufacturing and materials science applications. LS-DYNA has been in development since 1976.

| | |
|---|---|
| MD5 | 1d2f32c57ae2f2013f513d342925e972 |
| SHA1 | 2fa28ef1c6744bdc2021abd4048eefc777dccf22 |
| SHA256 | 5966513a12a5601b262c4ee4d3e32091feb05b666951d06431c30a8cece83010 |
| File Size | 5,225,591 bytes |
| Link time | 2003-10-24 16:34:57 UTC |
| File Type | PE32 executable for MS Windows 4.00 (console), Intel i386, 7 sections |

### Patch Target Candidate 2: PKPM Software Suite

Practical Structural Design and Construction Software (PKPM) is a structural engineering CAD software suite widely used in China for building design. The suite comprises multiple executable modules covering the full lifecycle of structural building design, from structural layout and concrete shear design for beams and columns to seismic, wind, and load analysis for high-rise buildings.

PKPM's core analysis engine, SATWE (Space Analysis of Tridimensional Wired Elements), handles tridimensional structural analysis across floors, beams, columns, walls, and frames. PKPM sees extensive use in Chinese civil engineering.

#### PKPM Concrete Code Shear Design Module

| | |
|---|---|
| MD5 | af4461a149bfd2ba566f2abefe7dcde4 |
| SHA1 | 586edef41c3b3fba87bf0f0346c7e402f86fc11e |
| SHA256 | 09ca719e06a526f70aadf34fb66b136ed20f923776e6b33a33a9059ef674da22 |
| File Size | 7716864 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 6 sections |
| Link Time | 2011-08-26 10:58:17 UTC |

#### PKPM Building Structure CAD Modules

| | |
|---|---|
| MD5 | 49a8934ccd34e2aaae6ea1e6a6313ffe |
| SHA1 | 3ce5b358c2ddd116ac9582efbb38354809999cb5 |
| SHA256 | 8b018452fdd64c346af4d97da420681e2e0b55b8c9ce2b8de75e330993b759a0 |
| File Size | 11849728 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 4 sections |
| Link Time | 2005-12-01 08:35:46 UTC |
| MD5 | e0c10106626711f287ff91c0d6314407 |
| SHA1 | 650fc6b3e4f62ecdc1ec5728f36bb46ba0f74d05 |
| SHA256 | 06361562cc53d759fb5a4c2b7aac348e4d23fe59be3b2871b14678365283ca47 |
| File Size | 16355328 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 5 sections |
| Link Time | 2012-07-07 08:47:11 UTC |

**PKPM SATWE Structural Analysis Engine**

| | |
|---|---|
| MD5 | 2717b58246237b35d44ef2e49712d3a2 |
| SHA1 | d475ace24b9aedebf431efc68f9db32d5ae761bd |
| SHA256 | bd04715c5c43c862c38a4ad6c2167ad082a352881e04a35117af9bbfad8e5613 |
| File Size | 9908224 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 6 sections |
| Link Time | 2011-01-12 06:37:39 UTC |
| MD5 | daea40562458fc7ae1adb812137d3d05 |
| SHA1 | 1ce1111702b765f5c4d09315ff1f0d914f7e5c70 |
| SHA256 | da2b170994031477091be89c8835ff9db1a5304f3f2f25344654f44d0430ced1 |
| File Size | 8454144 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 7 sections |
| Link Time | 2012-11-29 03:10:12 UTC |
| MD5 | 2740a703859cbd8b43425d4a2cacb5ec |

| | |
|---|---|
| SHA1 | ca665b59bc590292f94c23e04fa458f90d7b20c9 |
| SHA256 | aeaa389453f04a9e79ff6c8b7b66db7b65d4aaffc6cac0bd7957257a30468e33 |
| File Size | 16568320 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 5 sections |
| Link Time | 2014-12-30 03:23:43 UTC |
| MD5 | ebff5b7d4c5becb8715009df596c5a91 |
| SHA1 | 829f8be65dfe159d2b0dc7ee7a61a017acb54b7b |
| SHA256 | 37414d9ca87a132ec5081f3e7590d04498237746f9a7479c6b443accee17a062 |
| File Size | 8089600 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 6 sections |
| Link Time | 2009-04-22 01:46:46 UTC |
| MD5 | cb66a4d52a30bfcd980fe50e7e3f73f0 |
| SHA1 | e6018cd482c012de8b69c64dc3165337bc121b86 |
| SHA256 | 66fe485f29a6405265756aaf7f822b9ceb56e108afabd414ee222ee9657dd7e2 |
| File Size | 9219072 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 8 sections |
| Link Time | N/A |

**Additional PKPM CAD files**

| | |
|---|---|
| MD5 | 075b4aa105e728f2b659723e3f36c72c |
| SHA1 | 145ef372c3e9c352eaaa53bb0893749163e49892 |
| SHA256 | c11a210cb98095422d0d33cbd4e9ecc86b95024f956ede812e17c97e79591cfa |
| File Size | 6852608 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 6 sections |
| Link Time | 2012-06-18 10:01:54 UTC |
| MD5 | cf859f164870d113608a843e4a9600ab |
| SHA1 | 952ed694b60c34ba12df9d392269eae3a4f11be4 |

| | |
|---|---|
| SHA256 | 7e00030a35504de5c0d16020aa40cbaf5d36561e0716feb8f73235579a7b0909 |
| File Size | 8392704 bytes |
| File Type | PE32 executable for MS Windows 4.00 (GUI), Intel i386, 6 sections |
| Link Time | 2012-11-29 03:10:12 UTC |

### Candidate 3: MOHID Software Suite

Modelo Hidrodinâmico (Portuguese for “Hydrodynamic Model” or MOHID) is an open-source water modeling system developed by MARETEC (Marine and Environmental Technology Research Center) at the Instituto Superior Técnico in Lisbon, Portugal. The software is used for marine and coastal water modeling, covering hydrodynamics, water quality simulation, sediment transport, oil spill modeling, and Lagrangian particle tracking.

At this time, we cannot definitively identify the target and welcome contributions from the broader research community to aid understanding of the intended effects of attacking this software.

| | |
|---|---|
| MD5 | f4dbbb78979c1ee8a1523c77065e18a5 |
| SHA1 | 9e089a733fb2740c0e408b2a25d8f5a451584cf6 |
| SHA256 | e775049d1ecf68dee870f1a5c36b2f3542d1182782eb497b8ccfd2309c400b3a |
| File Size | 5443584 bytes |
| File Type | PE32 executable for MS Windows 4.00 (console), Intel i386, 3 sections |
| Link Time | 2002-10-18 09:29:54 UTC |

## Indicators of Compromise

| | |
|---|---|
| Name | fast16.sys |
| MD5 | 0ff6abe0252d4f37a196a1231fae5f26 |
| SHA1 | 92e9dcaf7249110047ef121b7586c81d4b8cb4e5 |
| SHA256 | 07c69fc33271cf5a2ce03ac1fed7a3b16357aec093c5bf9ef61fbfa4348d0529 |
| Name | connotify.dll |
| MD5 | 410eddfc19de44249897986ecc8ac449 |
| SHA1 | 675cb83cec5f25ebbe8d9f90dea3d836fcb1c234 |
| SHA256 | 8fcb4d3d4df61719ee3da98241393779290e0efcd88a49e363e2a2dfbc04dae9 |
| Name | svcmgmt.exe |

| | |
|---|---|
| MD5 | dbe51eabebf9d4ef9581ef99844a2944 |
| SHA1 | de584703c78a60a56028f9834086facd1401b355 |
| SHA256 | 9a10e1faa86a5d39417cae44da5adf38824dfb9a16432e34df766aa1dc9e3525 |

## YARA Rules

```
import "pe"

rule apt_fast16_carrier {
    meta:
        author = "SentinelLABS/vk"
        date = "2025-04-07"
        description = "Catches fast16 carrier, its Lua payload, and plaintext variants"
        hash = "9a10e1faa86a5d39417cae44da5adf38824dfb9a16432e34df766aa1dc9e3525"
    strings:
        $lua_magic = { 1B 4C 75 61 }


        $s1 = "build_wormlet_table"
        $s2 = "unpropagate"
        $s3 = "worm_install_failure_action"
        $s4 = "implant_install_failure_action"
        $s5 = "scm_wormlet_propagate_system"
        $s6 = "scm_wormlet_install"
        $s7 = "scm_wormlet_init"
        $s8 = "scm_copy_payload"
        $s9 = "get_logged_on_user"
        $s10 = "logged_on_program"
        $s11 = "phase_1_prop_delay"
        $s12 = "connotify_pipename"
        $s13 = "cndll_internal_name"
        $s14 = "connotify_provider_key"
        $s15 = "check_implant_reg_values"
        $s16 = "set_implant_reg_values"
        $s17 = "install_implant"
        $s18 = "implant_installed"
        $s19 = "implant_internal_name"
        $s20 = "implant_files"
        $s21 = "implant_owner"
        $s22 = "install_worm"
        $s23 = "start_worm"
        $s24 = "implant_install_failure_action"
        $s25 = "worm_install_failure_action"
        $s26 = "ok_to_propagate"
```

```
        $s27 = "no_firewall_check"
        $s28 = "scm_wormlet"
        $s29 = "implant_install_failure_action"
        $s30 = "worm_install_failure_action"


        $e1 = { 98 18 A1 94 24 E3 A2 4C  61 C8 AE 04 DC 4E 03 CD 0D 9D F0 }
        $e2 = { E8 76 53 6D D4 B9 6E 28  6C 5D C2 }
        $e3 = { 7D B7 14 73 F0 C0 4D 53  BB F7 0A 4A 3A 63 05 92  EC 0A 11 BC 22 59 99 05  72 05 19 
        $e4 = { 88 5F 1B E4 45 56 75 4B  A5 3D 19 0B 3F 30 5A 85  E2 BD D0 E7 1C 13 D0 1D  BD D8 CF 
        $e5 = { 88 1E 54 4E 00 C1 EF 79  AA AD 9F 50 27 B5 B8 4C  32 06 D2 7B 32 E3 AF D6  DC D2 BB 
        $e6 = { 39 F9 BC E9 27 70 C4 3E  04 2A 7D E1 68 67 B7 ED  D4 41 6A }
        $e7 = { 13 FC 24 20 1F 20 74 1B  E5 5F 59 56 D7 61 3E BD }
        $e8 = { EF 94 49 63 33 41 62 F2  26 A6 48 DE 6D 7B A4 CF }
        $e9 = { 36 5F 5E E5 C1 1A 17 6A  4E B9 94 52 1B DC C6 60  CA C7 }
        $e10 = { B3 9C A3 F1 12 CC 52 74  34 5F 87 43 32 21 36 7B 2A }

        $rk1 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Symantec\\InstalledApps"
        $rk2 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Sygate Technologies, Inc.\\Sygate Personal Firewall"
        $rk3 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\TrendMicro\\PFW"
        $rk4 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Zone Labs\\TrueVector"
        $rk5 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\F-Secure"
        $rk6 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Network Ice\\BlackIce"
        $rk7 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\McAfee.com\\Personal Firewall"
        $rk8 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\ComputerAssociates\\eTrust EZ Armor"
        $rk9 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\RedCannon\\Fireball"
        $rk10 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Kerio\\Personal Firewall 4"
        $rk11 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\KasperskyLab\\InstalledProducts\\Kaspersky Anti-Hacke
        $rk12 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Tiny Software\\Tiny Firewall"
        $rk13 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\Look n
        $rk14 = "HKEY_CURRENT_USER\\SOFTWARE\\Soft4Ever"
        $rk15 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Norman Data Defense Systems"
        $rk16 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Agnitum\\Outpost Firewall"
        $rk17 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Panda Software\\Firewall"
        $rk18 = "HKEY_LOCAL_MACHINE\\SOFTWARE\\InfoTeCS\\TermiNET"

        $c1 = { 86 3A D6 02 }
        $c2 = { 01 E1 F5 05 }

        $code1 = { 8B 00
        2D 2F 34 21 33
        }

        $stor1 = { CC 00 00 00 05 00 00 00 66 69 6C 65 00 CD 00 00 00 }
    condition:
        ( uint16(0)==0x5a4d and filesize < 10MB and (
        ( 3 of ($s*) ) or
```

```
        ( 12 of ($rk*) ) or
        ( any of ($e*) ) or
        ( all of ($c*) and @c2-@c1 < 0x100 ) or
        ( $code1 ) or
        ( $stor1 )) ) or
        ( $lua_magic and 7 of ($s*) )
}
```

```
rule apt_fast16_driver {
    meta:
        author = "SentinelLABS/vk"
        last_modified = "2026-04-15"
        description = "Catches fast16 driver or related project files"
        hash = "07c69fc33271cf5a2ce03ac1fed7a3b16357aec093c5bf9ef61fbfa4348d0529"
    strings:
        $a1 = "@(#)foo.c : "
        $a2 = "@(#)par.h : "
        $a3 = "@(#)pae.h : "
        $a4 = "@(#)fao.h : "
        $a5 = "@(#)uis.h : "
        $a6 = "@(#)ree.h : "
        $a7 = "@(#)fir.h : "
        $a8 = "@(#)fir.c : "
        $a9 = "@(#)par.h : "
        $a10 = "@(#)pae.h : "
        $a11 = "@(#)fao.h : "
        $a12 = "@(#)uis.h : "
        $a13 = "@(#)ree.h : "
        $a14 = "@(#)fir.h : "
        $a15 = "@(#)myy.h : "
        $a16 = "@(#)fic.h : "
        $a17 = "@(#)ree.h : "
        $a18 = "@(#)ree.c : "
        $dev1 = "\\Device\\fast16"
        $dev2 = "\\??\\fast16"
        $pdb1 = "C:\\buildy\\"
        $pdb2 = "driver\\fd\\i386\\fast16.pdb"
        $devtype = { 68 7C A5 00 00 } // push 0A57Ch ; DeviceType
        $api1 = {50 C6 45 D4 16 C6 45 D5 2B C6 45 D6 12 C6 45 D7 3F C6 45 D8 3F C6 45 D9 3C C6 45 DA
        $api2 = {C6 45 A8 16 C6 45 A9 2B C6 45 AA 12 C6 45 AB 3F C6 45 AC 3F C6 45 AD 3C C6 45 AE 30
        $api3 = {C6 45 E4 16 C6 45 E5 2B C6 45 E6 15 C6 45 E7 21 C6 45 E8 36 C6 45 E9 36 C6 45 EA 03
        $api4 = {C6 45 C0 16 C6 45 C1 2B C6 45 C2 15 C6 45 C3 21 C6 45 C4 36 C6 45 C5 36 C6 45 C6 03
    condition:
        filesize < 10MB and
        ( uint16(0)==0x5a4d and
        ( ( 2 of ($pdb*) ) or
```

```
        ( $pdb1 and 1 of ($a*) ) or
        (
        pe.machine == pe.MACHINE_I386 and
        pe.subsystem == pe.SUBSYSTEM_NATIVE) or
        any of ($api*) or
        2 of ($dev*))) or
        ( 6 of ($a*))
}
```

```
rule clean_fast16_patchtarget {
    meta:
        author = "SentinelLABS/vk"
        last_modified = "2026-04-15"
        description = "Detects fast16 patch target software (most probably clean)"
        hash = "8fcb4d3d4df61719ee3da98241393779290e0efcd88a49e363e2a2dfbc04dae9"
    strings:
        $el0 = { 48 89 84 24 9C 00 00 00 4B 0F 8F 79 FF FF FF 00 }
        $el10 = { D8 E1 D9 5D FC D9 04 00 }
        $el12 = { 55 8B EC 83 EC 14 53 56 57 8B 3D ?? ?? ?? ?? 8B 0D 00 }
        $el13 = { 89 4D C8 8B FB 8B C8 00 }
        $el14 = { 8B 4C 24 0C 8B 01 83 F8 63 00 }
        $el16 = { 39 2D ?? ?? ?? ?? 0F 84 F4 00 00 00 8B 35 ?? ?? ?? ?? 2B 35 }
        $el2 = { 7C 02 89 C6 89 35 ?? ?? ?? ?? 89 B4 24 D0 }
        $el23 = { 83 3D ?? ?? ?? ?? 00 0F 84 70 BD FF FF 00 }
        $el25 = { BE 07 00 00 00 BF 04 00 00 00 BB 02 00 00 00 00 }
        $el26 = { 8B 4D 10 C1 E2 04 8B 19 83 EA 30 8B CB 49 }
        $el28 = { 8D 1D ?? ?? ?? ?? 52 8D 05 ?? ?? ?? ?? 51 8D 15 ?? ?? ?? ?? 8D 0D ?? ?? ?? ?? 53 5
        $el3 = { 0F 8F A5 00 00 00 A1 ?? ?? ?? ?? 83 F8 14 7D 0D }
        $el30 = { 8B 5D B0 0F 85 ?? ?? ?? ?? 8D 34 9D ?? ?? ?? ?? 8D 14 9D 00 0F 8E 1B 03 00 00 D9 0
        $el31 = { 8B 45 44 6B 00 04 D9 05 ?? ?? ?? ?? D8 B0 }
        $el32 = { E9 7E 04 00 00 8B 74 24 1C 8B 54 24 14 85 }
        $el33 = { 83 39 63 0F 85 21 03 00 00 8B EE 85 F6 0F }
        $el34 = { 85 DB 8B 55 D4 75 2C 89 35 00 }
        $el36 = { 75 18 8D 35 ?? ?? ?? ?? 56 8D 3D 00 }
        $el37 = { 8D 1D ?? ?? ?? ?? 52 8D 05 ?? ?? ?? ?? 51 8D 15 ?? ?? ?? ?? 8D 0D ?? ?? ?? ?? 53 5
        $el39 = { D8 34 85 ?? ?? ?? ?? 8B 44 ?? ?? 8B CA 00 }
        $el4 = { 8B 5D 0C 8B 55 08 8B 36 8B 00 }
        $el40 = { 8D 04 BD ?? ?? ?? ?? 03 DF 00 }
        $el41 = { 8B EE 85 F6 0F 8E ?? ?? ?? ?? 8D 1C BD 00 }
        $el42 = { D9 04 9D ?? ?? ?? ?? 83 ED 04 05 10 00 00 00 D8 0D 00 }
        $el43 = { 75 2C 89 35 ?? ?? ?? ?? 89 05 ?? ?? ?? ?? 89 15 }
        $el45 = { 89 55 F4 8B F9 8B D3 03 FB C1 E2 02 89 35 }
        $el46 = { 40 23 72 65 63 24 65 69 69 6E 20 2E 30 24 D9 5D 00 D9 03 D8 0D ?? ?? ?? ?? D8 0D 0
        $el49 = { DF E0 F6 C4 41 A1 ?? ?? ?? ?? 74 5A }
        $el51 = { FF 35 ?? ?? ?? ?? E8 ?? ?? ?? ?? 9D D9 E0 D9 1D ?? ?? ?? ?? 8B 4C }
        $el53 = { 6A 46 68 ?? ?? ?? ?? E8 ?? ?? ?? ?? 6A 03 }
```

```
        $el56 = { D8 05 ?? ?? ?? ?? D9 55 00 9C }
        $el59 = { C2 08 00 A1 ?? ?? ?? ?? 8B 0C 85 ?? ?? ?? ?? 89 0E 00 }
        $el6 = { 83 EC 04 53 E8 ?? ?? ?? ?? EB 09 83 EC 04 53 00 }
        $el61 = { D8 1D ?? ?? ?? ?? DF E0 F6 C4 41 B8 00 00 00 00 75 05 B8 01 00 00 00 85 C0 74 11 6A
        $el63 = { 2B DA 89 3C 03 83 3D 00 }
        $el68 = { D9 5D C0 8B 4D C0 D9 45 E0 89 0E 00 }
        $el70 = { 8B 05 ?? ?? ?? ?? 8B 0D ?? ?? ?? ?? 0F 85 7E 00 00 00 0F AF 15 00 }
        $el73 = { B9 01 00 00 00 C1 E7 02 8B BF ?? ?? ?? ?? 8B D7 85 FF 8B 55 30 8B 45 30 D8 C9 8B 7
        $el75 = { 2B FB 8B DE C1 E3 02 89 7D A0 03 5D A0 8B 03 F7 F7 DB 0C 02 89 35 }
        $el80 = { 0F 0F 94 C0 23 C3 33 D2 }
        $el81 = { 8B 55 30 8B 75 2C D8 C9 8B 45 30 00 }
        $el83 = { DD 05 ?? ?? ?? ?? 8B 05 ?? ?? ?? ?? 8B 15 ?? ?? ?? ?? 0F AF 05 ?? ?? ?? ?? 8B 1D ?
        $el89 = { 68 28 00 00 00 57 E8 ?? ?? ?? ?? 8B 1D ?? ?? ?? ?? 8B 35 ?? ?? ?? ?? 0F AF 1D ?? ?
        $el94 = { 8B 75 38 8B 4D 34 D8 C9 8B 00 }
        $el96 = { 8B 55 88 8B 5D B0 83 7D 84 01 }
        $el97 = { 55 8B EC 83 EC 2C 33 D2 53 56 57 8B }
        $el99 = { 55 8B EC 83 EC 2C B9 46 00 00 00 53 56 57 8B 00 }
    condition:
        filesize < 20MB and
        uint16(0) == 0x5A4D and
        2 of them
}
```

```
rule apt_fast16_patch {
        meta:
                author = "SentinelLABS/vk"
                last_modified = "2026-04-15"
                description = "Detects the fast16 patch code. May be present in statically patched f
                hash = "0ff6abe0252d4f37a196a1231fae5f26"
        strings:
                $p1 = { 55 88 50 53 52 51 8D 64 24 94 DD 34 24 51 E8 ?? ?? ?? ?? 59 81 E9 14 00 00 0
                $p2 = { 59 81 E9 EE 00 00 00 6A 02 BB B4 05 00 00 01 CB C6 03 EB 43 C6 03 15 8B 44 2
                $p3 = { 50 53 52 51 E8 ?? ?? ?? ?? 59 81 E9 78 01 00 00 D9 99 C4 0F 00 00 8D 64 24 9
        condition:
                any of them
}
```

---

Source: https://www.sentinelone.com/labs/fast16-mystery-shadowbrokers-reference-reveals-high-precision-software-sabotage-5-years-before-stuxnet/