Discovering Splinter: A First Look at a New Post-Exploitation Red Team Tool

By Dominik Reichel

Published: 2024-09-19 · Archived: 2026-04-05 16:43:22 UTC

Executive Summary

This article discusses the discovery of a new post-exploitation red team tool called Splinter that we found on customer systems using Advanced WildFire's memory scanning tools. Penetration testing toolkits and adversary simulation frameworks are often useful for identifying potential security issues in a company's network. However, these tools can sometimes end up in the hands of criminals, highlighting the need for continuous tracking and detection of them.

Palo Alto Networks customers are better protected from the Splinter post-exploitation tool through Advanced WildFire with its different memory analysis features. The Advanced WildFire machine-learning models and analysis techniques have been reviewed and updated in light of the IoCs shared in this research. Advanced WildFire classifies the Splinter malware samples discussed in this article as malicious.

Cortex XDR and XSIAM help detect and block known samples, and Behavioral Threat Protection monitors for post-exploitation activity.

If you think you might have been compromised or have an urgent matter, contact the Unit 42 Incident Response team.

Introduction to Splinter

Earlier this year, our Advanced WildFire memory scanning tools discovered a new post-exploitation red team tool on a customer system. By searching our sample telemetry database, we discovered that several customers were affected.

Several string artifacts in the samples, as well as the collection of features, make it evident that Splinter is a red team tool. This tool's name is its internal project name, which was left behind in a debug artifact. We don't yet know who developed Splinter – we have only a few hints that don't lead to a significant conclusion.

When used responsibly, penetration testing toolkits and adversary simulation frameworks can significantly improve a company's security. Their primary purpose is to identify potential vulnerabilities in a company's network before an attacker exploits them.

Many of these toolkits include post-exploitation capabilities. Post-exploitation tools are often custom developed with the goal of expanding initial access gained and simulating long-term access on a target system.

The most well-known example of this sort of toolkit is Cobalt Strike. Although it's proprietary software that only legal clients can acquire, sometimes it ends up in the hands of criminals.

During our analysis, we have not identified threat actor activity associated with the Splinter tool set.

Technical Analysis

Splinter is developed in Rust, a relatively new programming language that's recommended for developing memory-safe software. However, it has densely layered runtime code, which amounts for up to 99% of a program's code. This density makes analysis a real challenge for malware reverse engineers.

The sample found on a customer system (SHA-256: 1962cef10cf737300d04a23139122abcc8e8803e54dfcb63054140fbe549bed0) is a 64-bit executable that was linked with debug information that has the following PDB path:

- C:\gitlab-runner\builds_fUzhMf8i\0\h3upperbounds\red-team\implant\splinter_core\target\release\deps\implant_exe.pdb

As this file path shows, the project name of this post-exploitation tool is Splinter. A single sample is defined as an implant, a typical term for a red-team post-exploitation tool. Other samples are compiled as DLLs with a PDB path ending with implant_dll.pdb.

While Rust samples are typically large, ranging from a few hundred kilobytes to a few megabytes, a typical Splinter sample is exceptionally large at around 7 MB. This is mostly due to its use and of large external libraries that are statically linked into the file. These are referred to as crates in Rust terminology.

The sample uses the following crates:

- indexmap (2.2.3)
- futures-channel (0.3.30)
- tracing-core (0.1.32)
- matchers (0.1.0)
- hyper-rustls (0.24.2)
- regex-syntax (0.6.29, 0.8.2)
- tokio-util (0.7.10)
- hashbrown (0.14.3, 0.14.0)
- tracing-subscriber (0.3.18)
- tokio-rustls (0.24.1)
- parking_lot (0.12.1)
- once_cell (1.19.0)
- sharded-slab (0.1.7)
- socket2 (0.5.5)
- windows-core (0.51.1)
- sct (0.7.1)
- url (2.5.0)

- percent-encoding (2.3.1)
- lazy_static (1.4.0)
- smallvec (1.13.1)
- serde (1.0.196)
- spin (0.9.8)
- tinyvec (1.6.0)
- ring (0.17.7)
- regex-automata (0.4.5, 0.1.10)
- backtrace (0.3.69)
- crossbeam-channel (0.5.11)
- serde_json (1.0.113)
- anyhow (1.0.79)
- ipnet (2.9.0)
- encoding_rs (0.8.33)
- reqwest (0.11.24)
- rustc-demangle (0.1.23)
- want (0.3.1)
- tracing-appender (0.2.3)
- mio (0.8.10)
- unicode-normalization (0.1.22)
- rustls-pemfile (1.0.4)
- mime (0.3.17)
- parking_lot_core (0.9.9)
- bytes (1.5.0)
- httparse (1.8.0)
- futures-util (0.3.30)
- thread_local (1.1.7)
- rustls-webpki (0.101.7)
- time (0.3.34)
- h2 (0.3.24)
- untrusted (0.9.0)
- rmp-serde (1.1.2)
- tracing-log (0.2.0)
- futures-core (0.3.30)
- regex (1.10.3)
- log (0.4.20)
- idna (0.5.0)
- uuid (1.7.0)
- tokio (1.36.0)
- http (0.2.11)
- base64 (0.21.7)
- slab (0.4.9)

- hyper (0.14.28)
- rustls (0.21.10)

Like many other post exploitation tools, Splinter uses a configuration data structure in JSON format that contains the necessary information for its operations. The data structure is internally named ImplantConfig and contains the following information:

- id (correlation_id in older samples): Implant ID [string]
- weakness_uuid: Unknown ID (probably related to an exploited vulnerability) [string]
- endpoint_uuid: Targeted endpoint ID [string]
- is_test_implant: Whether the file is a test sample [boolean]
- c2_server_address: Command and control (C2) server address [string]
- c2_port: C2 server port [int]
- c2_user: C2 username [string]
- c2_password: C2 user password [string]
- log_path: Path of log file [string]
- log_env: Log level [string]

As an example, our Splinter sample contains the following data:

```
1  {
2  "id":"fd06a788-75e9-4f27-b5f5-ae8ea636dba2",
3  "weakness_uuid":"00000000-0000-0000-0000-000000000000",
4  "endpoint_uuid":"00000000-0000-0000-0000-000000000000",
5  "is_test_implant":false,
6  "c2_server_address":"192.168.5[.]151",
7  "c2_port":28069,
8  "c2_user":"BrqUjhYhvRwkKpyQZZKf",
9  "c2_password":"JjAxsdEPZqRJuFebHyKQ",
10 "log_path":null,
11 "log_env":null
12 }
13
14
```

```
15
16
17
18
19
20
21
22
23
```

Upon execution, the sample parses the configuration data and it uses the network information to connect to the C2 server using HTTPS with the login credentials. Splinter implants are controlled by a task-based model, which is common among post-exploitation frameworks. It obtains its tasks from the C2 server the attacker has defined. Splinter tasks have the following post-exploitation features:

- Execute a Windows command
- Execute a module via remote process injection
- Upload a file from the victim's system to the attacker's server
- Drop a file from the attacker's server to the victim's system
- Gather information from a certain cloud service account
- Self-delete

Splinter uses the classic process injection method as an option for running additional modules.

Figure 1 shows thread creation in a remote process that runs a PE loader shellcode that in turn executes the payload. Both the PE loader and the payload are written to the remote process defined by the attacker.

```
330  *a5 = p_buffer_with_payload;
331  mem_alloc_type = create_mem_alloc_type(MEM_RESERVE, MEM_COMMIT);
332  p_remote_buffer = VirtualAllocEx(hProcess, 0LL, 1040uLL, mem_alloc_type, PAGE_EXECUTE_READWRITE);
333  p_remote_buffer_with_payload = p_remote_buffer;
334  if ( !p_remote_buffer )
335    goto LABEL_54;
336  if ( !WriteProcessMemory(hProcess, p_remote_buffer, a5, 1040uLL, 0LL) )
337    goto LABEL_57;
338  ThreadId[0] = 0;
339  hRemoteThread = CreateRemoteThread(
340                    hProcess,
341                    0LL,
342                    0LL,
343                    (LPTHREAD_START_ROUTINE)remote_pe_loader_shellcode,
344                    p_remote_buffer_with_payload,
345                    9u,
346                    ThreadId);
347  if ( check_remote_thread_handle(&hRemoteThread) )
348  {
349    v42 = get_last_error();
```

Figure 1. Remote process injection to run additional payloads.

Splinter uses the following URL paths on the attacker's C2 server to synchronize tasks, maintain a heartbeat connect, and download or upload files:

- /implant/task_created_events: Used for task synchronization
- /implant/task_completed_events: Used for task status processing
- /implant/files/: Used to download/upload files
- /implant/heartbeat: Used to check if the implant is alive and has a connection to the C2 server

All network communication is encrypted with HTTPS.

Conclusion

In this article, we reveal Splinter, a new post-exploitation red team tool that we have found on several client systems. It has a standard set of features commonly found in penetration testing tools and its developer created it using the Rust programming language. While Splinter is not as advanced as other well-known post-exploitation tools like Cobalt Strike, it still presents a potential threat to organizations if it is misused.

This discovery emphasizes the increasing number of red-teaming tools available. There is therefore an increasing variety of ways that an organization's environment could reflect threat actor-style activity. The increasing variety underscores the importance of staying up to date on prevention and detection capabilities, since criminals are likely to adopt any techniques that are effective for compromising organizations.

Palo Alto Networks customers receive better protection from this threat through Advanced WildFire. The Advanced WildFire machine-learning models and analysis techniques have been reviewed and updated in light of the IoCs shared in this research.

Cortex XDR and XSIAM help detect and block known samples, and Behavioral Threat Protection monitors for post-exploitation activity.

If you think you may have been compromised or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America Toll-Free: 866.486.4842 (866.4.UNIT42)
- EMEA: +31.20.299.3130
- APAC: +65.6983.8730
- Japan: +81.50.1790.0200

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

Sample Hash (SHA-256)

- 1962cef10cf737300d04a23139122abcc8e8803e54dfcb63054140fbe549bed0

Source: https://unit42.paloaltonetworks.com/analysis-pentest-tool-splinter/