ATREDIS PARTNERS

Open Technology Fund INVISV MASQUE Platform Security Assessment Security Assessment Report

Project Team:
Technical Testing: James Cook
Technical Editing: Matt South
Project Management: Holly Leitru

Prepared for Open Technology Fund, Inc.
September 4, 2024 (version 1.2)

Table of Contents

Engagement Overview

Assessment Components and Objectives

Open Technology Fund, Inc. (“Open Technology Fund”) recently engaged Atredis Partners (“Atredis”) to perform a Platform Security Assessment of the INVISV MASQUE platform. Objectives included validation that INVISV MASQUE services were developed and deployed with security best practices in mind, and to obtain third party validation that any significant vulnerabilities present in INVISV MASQUE platform were identified for remediation.

Testing was performed from June 28 through July 12, 2024 by James Cook of the Atredis Partners team with Holly Leitru 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
OPEN TECHNOLOGY FUND INVISV MASQUE PLATFORM SECURITY ASSESSMENT	
Assessment Targets	▪ INVISV MASQUE ▪ Implementation of the IETF MASQUE tunneling protocol ▪ MASQUE has been updated and ratified by the IETF after INVISV MASQUE was written ▪ INVISV MASQUE implements a draft version of the protocol ▪ Client library written in Go ▪ Provides the client-side functionality needed for running a multi-party relay service ▪ HTTP/2 implementation extends the Go standard library ▪ HTTP/3 implementation utilizes quic-go that was patched to implement MASQUE ▪ The patches have not been accepted by the quic-go upstream and INVISV’s quic-go fork is 347 commits behind at the time of this writing ▪ INVISV will pull upstream commits prior to start of the engagement ▪ Performs certificate pinning in both MASQUE and quic-go

COMPONENT	ENGAGEMENT TASKS
Assessment Tasks	▪ Source-Assisted Penetration Testing of INVISV MASQUE ▪ Automated and runtime application testing ▪ Protocol fuzzing and fault injection ▪ PoC generation and validation of findings ▪ Source code review
REPORTING AND ANALYSIS	
Analysis and Deliverables	▪ Status Reporting and Realtime Communication ▪ Comprehensive Engagement Deliverable ▪ Engagement Outbrief and Remediation Review

The ultimate goal of the assessment was to provide a clear picture of risks, vulnerabilities, and exposures as they relate to accepted security best practices, such as those created by the National Institute of Standards and Technology (NIST), Open Web Application Security Project (OWASP), or the Center for Internet Security (CIS). Augmenting these, Atredis Partners also draws on its extensive experience in secure development and in testing high-criticality applications and advanced exploitation.

Engagement Tasks

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

Network Protocol Analysis

With the objective of identifying scenarios where the integrity of trusted communications can be diminished or reduced, Atredis Partners reviewed network traffic using various packet flow analysis and packet capture tools to observe in-scope network traffic. Network communications were analyzed for the presence of cleartext communications or scenarios where the integrity of cryptographic communications can 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.

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 was obtained. To aid in this process, the assessment team engaged key stakeholders and members of the development team where possible to provide structured walkthroughs and interviews, helping the team rapidly gain an understanding of the application's design and development lifecycle.

Status Reporting and Realtime Communication

Atredis Partners scheduled regular status meetings with client representatives during the project and reported findings in realtime as soon as they were confirmed, via secure communication channels.

High Priority Issue Reporting

For Critical and High severity issues discovered during the engagement, Atredis delivered the details to the client once the vulnerability was confirmed. Atredis worked with the client's team to ensure understanding of the issue, the impact, and the proposed mitigation strategy.

Status Reports and Meetings

Atredis delivered, at a minimum, weekly status reports to the client, and (if requested), scheduled status meetings during the engagement. Standard Atredis status reports include project overview and tracking information such as percentage complete on each phase of the testing process. Reporting and meetings deliver relevant issues discovered that week as well as provide remediation guidance and additional information to the client team.

Executive Summary

Atredis Partners comprehensively evaluated the `masque` and `quic-go-upstream` source code repositories. This involved static code analysis and runtime testing, with the goal of identifying potential vulnerabilities, code quality issues, and runtime anomalies. The aim was to ensure the codebase's robustness, security, and performance, thereby enhancing the reliability and security of the Multiplexed Application Substrate over QUIC Encryption (MASQUE) protocol implementation.

Atredis Partners conducted static code analysis using industry-standard tools to scrutinize the source code for potential issues without executing the program. This analysis focused on identifying coding errors, security vulnerabilities, and adherence to coding standards.

Next, Atredis Partners performed runtime testing, which involved executing the packages implementing HTTP2 and HTTP3 in various scenarios to observe their behavior under different conditions. This testing aimed to uncover runtime errors and ensure the application's overall stability.

Key Conclusions

Overall, Atredis Partners found that the codebase is generally robust with no high or critical severity findings. Atredis Partners performed static code analysis and runtime testing of the Invisv-Privacy repositories `masque` commit (`587bcbc`) and `quic-go-upstream` commit (`66efcd1`).

While performing extensive manual and automated testing of the MASQUE implementation, Atredis Partners observed no critical runtime errors or crashes during testing. However, areas for improvement were identified, including the presence of logical expressions that are always true, the use of reserved words and package names as variables, unhandled errors, vulnerable dependencies, and deprecated functions.

Addressing the identified issues through code refactoring, error handling enhancement, dependency updates, and replacing deprecated functions will strengthen the `masque` and `quic-go-upstream` code quality, security, and maintainability, ensuring its readiness for production deployment and long-term success.

Platform Overview

The IETF MASQUE protocol is designed to enable the tunneling of arbitrary traffic through a MASQUE-supporting server using HTTP. This protocol generalizes the HTTP CONNECT method, creating multiple encrypted HTTP tunnels within a single connection. This makes it possible to tunnel various types of traffic, including TCP and UDP, through a single MASQUE connection. MASQUE can be used to enhance user privacy by tunneling traffic through a trusted server, masking the user's IP address and encrypting the data.

The Invisiv-Privacy MASQUE implementation is written in Go and includes sample applications to demonstrate its usage. For example, a Relay HTTP Proxy application locally presents a standard HTTP proxy interface and tunnels traffic through the MASQUE server.

The Invisiv-Privacy MASQUE implementation implements certificate pinning to enhance security by ensuring the client connects to a trusted server. Certificate pinning is a security mechanism that helps prevent man-in-the-middle (MITM) attacks by associating a host with its expected public key or certificate. This is achieved through the `VerifyPeerCertificate` field in the `tls.Config` struct.

Invisiv-Privacy utilizes Docker to manage an `h2o`[1] container to help with manual and automated testing. The h2o application is a HTTP server with support for HTTP/1, HTTP/2, and HTTP/3.

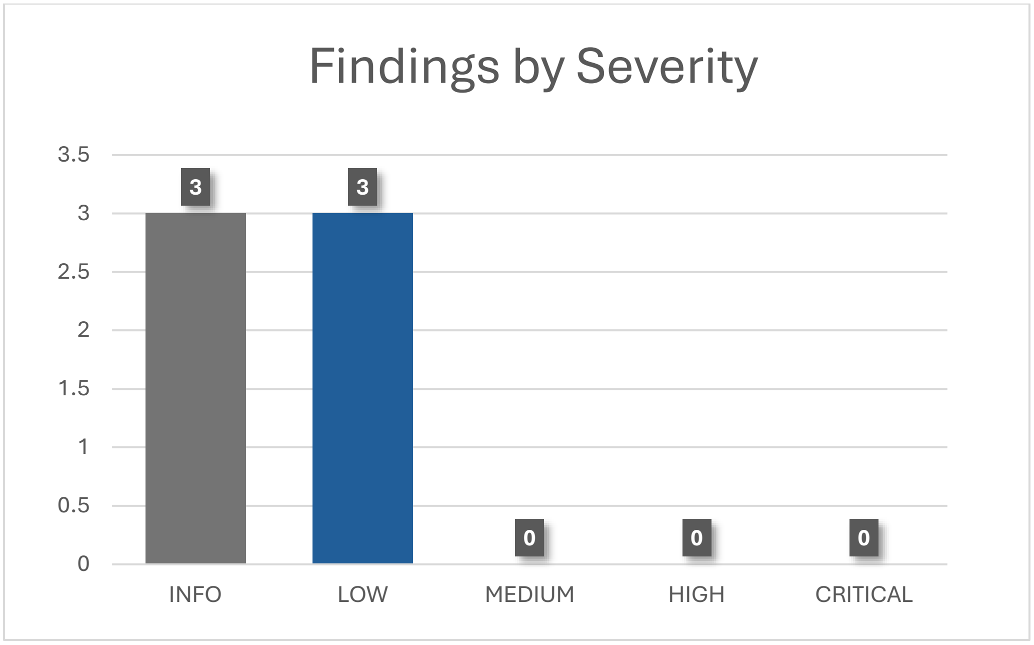
Findings Overview

In performing testing for this assessment, Atredis Partners identified **three (3) low** severity findings and **three (3) informational** findings. No high or critical severity findings were noted. As stated earlier, none of these issues constitute a potential for direct compromise.

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.
- **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.

[1] https://github.com/h2o/h2o



Remediation Tasks

In the case of this assessment, remediation tasks are not complex, and relate to minor code changes and updating dependencies to the latest versions.

As described elsewhere, no directly or indirectly exploitable conditions were noted, and as such, remediation tasks would be primarily driven by the objective of increasing overall application hardening and diminishing the impact of any potential future vulnerabilities that may arise in the platform.

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

CRITICAL	HIGH	MEDIUM	LOW	INFO
0	**0**	**0**	**3**	**3**

Findings Detail

FINDING NAME	SEVERITY	RETEST RESULTS
MASQUE: Unhandled Error	**LOW**	**REMEDIATED**
MASQUE: Vulnerable Dependency	**LOW**	**PARTIALLY REMEDIATED**
MASQUE: Use of Deprecated Function	**LOW**	**REMEDIATED**
MASQUE: Expression is Always True	**INFO**	**REMEDIATED**
MASQUE: Reserved Word Used as Variable	**INFO**	**REMEDIATED**
MASQUE: Package Name Used as Variable	**INFO**	**REMEDIATED**

MASQUE: Unhandled Error

SEVERITY: LOW

RETEST RESULTS: REMEDIATED

Finding Overview

While performing static code analysis of the `http2` package, Atredis Partners identified a potential unhandled error in the `http2\client.go` and `http2\conn.go` files.

Finding Detail

If the application encounters an error during the HTTP `ReadResponse` process, it then attempts to close the transport. However, transport may also return an error that is not currently handled. As shown in the following code snippet, the `tr.Close()` function returns an error that is not checked:

```
br := bufio.NewReader(tr)
resp, err := http.ReadResponse(br, nil)

if err != nil {
    tr.Close()
    return nil, fmt.Errorf("reading HTTP response from CONNECT-UDP to %s via proxy
%s failed: %v",
         addr, c.proxyAddr, err)
}
if resp.StatusCode != 200 {
    return nil, fmt.Errorf("proxy error from %s while dialing %s: %v",
         c.proxyAddr, addr, resp.Status)
}
```

Unhandled returned error

The following list of locations were identified in the `http2` package that contain unhandled errors:

- `http2\client.go:527`
- `http2\client.go:572`
- `http2\conn.go:63`
- `http2\conn.go:71`

Recommendation(s)

While it is unlikely that the identified functions will return an error, it is best practice to check the return value of the function and handle the error accordingly.

The following code snippet demonstrates how to handle this type of error:

```
if err := tr.Close(); err != nil {
    return nil, fmt.Errorf("failed closing transport %v\nreading HTTP response
from CONNECT-UDP to %s via proxy %s",
         err, addr, c.proxyAddr)
}
```

References

CWE-252: Unchecked Return Value:
https://cwe.mitre.org/data/definitions/252.html

Retest Results

During retesting, Atredis Partners confirmed the unhandled errors are now checked and logged.

```
if err := udp.doClose(); err != nil {
    c.logger.Error("Error from udp.doClose in decodeLoopUDP", "err", err)
}
```

Updated Error Handling

MASQUE Vulnerable Dependency

SEVERITY: LOW

RETEST RESULTS: PARTIALLY REMEDIATED

Finding Overview

While performing static code analysis of the `masque` package, Atredis Partners identified vulnerable dependencies in the `go.mod` file.

Finding Detail

The `golang.org/x/net` dependency is vulnerable to `CVE-2023-45288`, allowing an attacker to cause an HTTP/2 endpoint to read arbitrary amounts of header data. Next, the `github.com/quic-go/quic-go` dependency is vulnerable to `CVE-2024-22189`, allowing for an attacker to cause an endpoint to run out of memory by sending an extensive amount of `NEW_CONNECTION_ID` frames. Finally, the minimum required go version is vulnerable to `CVE-2024-24791`, allowing an attacker to cause a denial of service against an endpoint with `Expect: 100-continue` requests.

```
module github.com/invisv-privacy/masque

go 1.21.6

require (
    github.com/quic-go/quic-go v0.41.0
    github.com/stretchr/testify v1.8.4
    github.com/testcontainers/testcontainers-go v0.28.0
    github.com/testcontainers/testcontainers-go/modules/compose v0.28.0
    golang.org/x/net v0.21.0
    inet.af/netaddr v0.0.0-20230525184311-b8eac61e914a
)
```

Vulnerable Dependencies

Recommendation(s)

Atredis Partners recommends updating all dependencies with the latest versions where possible. The fix for `golang.org/x/net` was available in version `0.23.0`, the fix for `github.com/quic-go/quic-go` was available in version `0.42.0` and requiring the minimum version `go1.22.5` will remediate the `golang.org/x/net` vulnerability.

References

HTTP/2 CONTINUATION flood in net/http
https://pkg.go.dev/vuln/GO-2024-2687

Denial of service due to improper 100-continue handling in net/http
https://pkg.go.dev/vuln/GO-2024-2963

Close connection when an abnormally large number of frames are queued
https://nvd.nist.gov/vuln/detail/CVE-2024-22189

CWE-1395: Dependency on Vulnerable Third-Party Component
https://cwe.mitre.org/data/definitions/1395.html

Retest Results

During retesting, Atredis Partners confirmed the vulnerable dependency `github.com/quic-go/quic-go` and `golang.org/x/net` have been updated.

The minimum Go version has not changed, but at the time of retesting, the source code does not appear to expose this vulnerability.

```
module github.com/invisv-privacy/masque

go 1.21.6

require (
	github.com/quic-go/quic-go v0.42.0
	github.com/stretchr/testify v1.8.4
	github.com/testcontainers/testcontainers-go v0.28.0
	github.com/testcontainers/testcontainers-go/modules/compose v0.28.0
	golang.org/x/net v0.23.0
	inet.af/netaddr v0.0.0-20230525184311-b8eac61e914a
)
```

Updated Dependency

MASQUE: Use of Deprecated Function

SEVERITY: LOW

RETEST RESULTS: REMEDIATED

Finding Overview

The `DialTLS` function is used to create a new client connection to the server but was deprecated in Go 1.13 and later.

Finding Detail

While performing static code analysis of the `http2` package, Atredis Partners identified the use of `DialTLS` in `http2\client.go:351` and `http2\client.go:118`:

```
func (c *Client) dialProxyViaH2() (*gohttp2.Transport, error) {
    tr := &gohttp2.Transport{
         // Note that h2 Transport will bypass |network|, |addr|, and |cfg| and
         // return a default tls dialer to proxy w/ |proxyAddr|.
         DialTLS: func(network, addr string, cfg *tls.Config) (net.Conn, error) {
              t := c.makeTLSDialer()
              ctx, _ := makeTLSDialerContext(c.tlsTimeout)
              return t.DialContext(ctx, "tcp", c.proxyAddr)
         },
         AllowHTTP:          true,
         DisableCompression: true,
         WriteByteTimeout:   time.Second * 3,
         ReadIdleTimeout:    time.Second * 3,
         PingTimeout:        time.Second * 2,
    }
    return tr, nil
}
```

Function `DialTLS` set

Recommendation(s)

Atredis Partners recommends using the `DialContext` function instead of `DialTLS` function.

References

CWE-477: Use of Obsolete Function:
https://cwe.mitre.org/data/definitions/477.html

Go 1.14 Release Notes:
https://go.dev/doc/go1.14

Retest Results

During retesting, Atredis Partners confirmed the deprecated function `DialTLS` has been replaced with the supported `DialTLSContext`.

```
func (c *Client) dialProxyViaH2() (*gohttp2.Transport, error) {
    tr := &gohttp2.Transport{
         // Note that h2 Transport will bypass |network|, |addr|, and |cfg| and
         // return a default tls dialer to proxy w/ |proxyAddr|.
         DialTLSContext: func(ctx context.Context, network, addr string, cfg
*tls.Config) (net.Conn, error) {
              t := c.makeTLSDialer()
              return t.DialContext(ctx, "tcp", c.proxyAddr)
         },
         AllowHTTP:          true,
         DisableCompression: true,
         WriteByteTimeout:   time.Second * 3,
         ReadIdleTimeout:    time.Second * 3,
         PingTimeout:        time.Second * 2,
    }
    return tr, nil
}
```

Updated Function

MASQUE: Expression is Always True

SEVERITY: INFO

RETEST RESULTS: REMEDIATED

Finding Overview

While performing static code analysis of the `masque` package, Atredis Partners identified an expression that is always true in the `config.go` file.

Finding Detail

The `IsDisallowedPort` function in the `config.go` file contains an `if` statement that checks if the `disallowedPortsBitsetInitialized` variable is `False`. However, the variable is never updated resulting in the statement always returning true:

```
var disallowedPortsBitsetInitialized = false

func initDisallowedPortsBitset() {
    for _, port := range disallowedPorts {
         disallowedPortsBitset[port] = true
    }
}

// IsDisallowedPort returns true if the given destination port number is a value
that will be rejected by Fastly.
func IsDisallowedPort(dport uint16) bool {
    if !disallowedPortsBitsetInitialized {
         initDisallowedPortsBitset()
    }
    return dport <= MAX_DISALLOWED_PORT_NUM && disallowedPortsBitset[dport]
}
```

Variable `disallowedPortsBitsetInitialized` is always `false`

Recommendation(s)

There are two ways to remediate this issue. The first would be to remove the `if` statement. The second option would be to update the `disallowedPortsBitsetInitialized` variable so the check actually takes effect.

References

CWE-571: Expression is Always True:
https://cwe.mitre.org/data/definitions/571.html

Retest Results

During retesting, Atredis Partners confirmed the variable `disallowedPortsBitsetInitialized` is now set to `true` once the function `initDisallowedPortsBitset()` is called.

```
var disallowedPortsBitsetInitialized = false

func initDisallowedPortsBitset() {
    for _, port := range disallowedPorts {
         disallowedPortsBitset[port] = true
    }
    disallowedPortsBitsetInitialized = true
}

// IsDisallowedPort returns true if the given destination port number is a value
that will be rejected by Fastly.
func IsDisallowedPort(dport uint16) bool {
    if !disallowedPortsBitsetInitialized {
         initDisallowedPortsBitset()
    }
    return dport <= MAX_DISALLOWED_PORT_NUM && disallowedPortsBitset[dport]
}
```

Variable Properly Initialized

MASQUE: Reserved Word Used as Variable

SEVERITY: INFO

RETEST RESULTS: REMEDIATED

Finding Overview

While performing static code analysis of the `http2` package, Atredis Partners identified the use of the reserved word `len` as a variable name in the `http2\client.go` file.

Finding Detail

The reserved word `len` is used as a variable name in the `StreamDataToDatagramChunk` function located at `http2\client.go:398`:

```
func StreamDataToDatagramChunk(payload []byte, l int) ([]byte, int) {
    encode := make([]byte, 5)
    encode[0] = 0x00
    len := uint(l)
    if len > 63 {
        if len > 16383 {
            encode[1] = 0x80 | uint8(len>>24) // 4-byte length encoding
            encode[2] = uint8(len >> 16)
            encode[3] = uint8(len >> 8)
            encode[4] = uint8(len)
            encode = append(encode[:4:4], payload...)
            return encode, l + 5
        }
        encode[1] = 0x40 | uint8(len>>8) // 2-byte length encoding
        encode[2] = uint8(len)
        encode = append(encode[:3:3], payload...)
        return encode, l + 3
    }
    encode[1] = uint8(len) // 1-byte length encoding
    encode = append(encode[:2:2], payload...)
    return encode, l + 2
}
```

Reserved word used as variable name

Shadowing a reserved word can cause confusion and make the code harder to read and maintain, which can indirectly lead to logic errors.

Recommendation(s)

Atredis Partners recommends renaming the variable `len` to a different name to avoid shadowing the reserved word `len`. This makes the code clearer and prevents potential bugs related to variable shadowing.

References

CWE-710: Improper Adherence to Coding Standards:
https://cwe.mitre.org/data/definitions/710.html

Retest Results

During retesting, Atredis Partners confirmed the previous reserved word `len` has been changed to `dataLength`.

```
func StreamDataToDatagramChunk(payload []byte, l int) ([]byte, int) {
    encode := make([]byte, 5)
    encode[0] = 0x00
    dataLength := uint(l)
    if dataLength > 63 {
        if dataLength > 16383 {
            encode[1] = 0x80 | uint8(dataLength>>24) // 4-byte length encoding
            encode[2] = uint8(dataLength >> 16)
            encode[3] = uint8(dataLength >> 8)
            encode[4] = uint8(dataLength)
            encode = append(encode[:4:4], payload...)
            return encode, l + 5
        }
        encode[1] = 0x40 | uint8(dataLength>>8) // 2-byte length encoding
        encode[2] = uint8(dataLength)
        encode = append(encode[:3:3], payload...)
        return encode, l + 3
    }
    encode[1] = uint8(dataLength) // 1-byte length encoding
    encode = append(encode[:2:2], payload...)
    return encode, l + 2
}
```

Updated Reserved Word

MASQUE: Package Name Used as Variable

SEVERITY: INFO

RETEST RESULTS: REMEDIATED

Finding Overview

While performing static code analysis of the `http2` package, Atredis Partners identified the use of the package `url` as a variable name in the `http2\client.go` file.

Finding Detail

The `url` package is imported at the top of the `http2\client.go` file:

```
import (
    "bufio"
    "context"
    "crypto/tls"
    "errors"
    "fmt"
    "io"
    "log/slog"
    "net"
    "net/http"
    "net/url"
    "strings"
    "sync"
```

Package `url` imported

However, `url` is assigned as a variable shadowing the imported package name located at `http2\client.go:242`:

```
if addr[0] >= '0' && addr[0] <= '9' {
    // Golang mis-parses FQDNs with leading digits, so treat it like an IP.
    dst = "http://" + addr
    authority = addr
} else {
    url, err := url.Parse(addr)
    if err != nil {
        if addr[0] >= '0' && addr[0] <= '9' && strings.Contains(addr, ":") {
            // Golang mis-parses FQDNs with leading digits, so check for basic
format and pass.
            dst = addr
        } else {
            return nil, errors.New("an invalid destination addr: not a IPPort or
URL")
        }
    } else {
        dst = url.String()
```

Package `url` used as variable name

Shadowing an imported package name can cause confusion and make the code harder to read and maintain, which can indirectly lead to logic errors.

Recommendation(s)

Atredis Partners recommends renaming the variable `url` to a different name to avoid shadowing the imported `url` package. This makes the code clearer and prevents potential bugs related to variable shadowing.

References

CWE-710: Improper Adherence to Coding Standards:
https://cwe.mitre.org/data/definitions/710.html

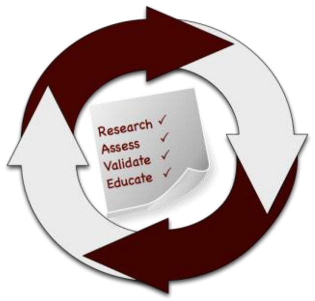
Retest Results

During retesting, Atredis Partners confirmed the previous variable `url` has been changed to `parsedURL`.

```
if addr[0] >= '0' && addr[0] <= '9' {
    // Golang mis-parses FQDNs with leading digits, so treat it like an IP.
    dst = "http://" + addr
    authority = addr
} else {
    parsedURL, err := url.Parse(addr)
    if err != nil {
        if addr[0] >= '0' && addr[0] <= '9' && strings.Contains(addr, ":") {
            // Golang mis-parses FQDNs with leading digits, so check for basic
format and pass.
            dst = addr
        } else {
            return nil, errors.New("an invalid destination addr: not a IPPort or
URL")
        }
    } else {
        dst = parsedURL.String()
```

Updated Variable Name

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 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 understanding 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 tool chains, and applicable testing artifacts, parses and normalizes these results, and presents an initial finding 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 our brief 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 tool chain 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.

Appendix II: Engagement Team Biographies

James Cook, Principal Research Consultant

James executes highly technical software security, network, and web application assessments and advanced red team assessments.

Experience

James has over 6 years of experience in the information security industry. James has developed and released numerous open-source security tools that are used by many information security professionals. He holds the Offensive Security Certified Professional (OSCP) certification and has performed a wide variety of security assessments including network penetration testing, application security assessments, full-scope red team engagements, adversarial simulation, and physical penetration testing

Prior to joining Atredis Partners, James performed network penetration tests as a Senior Security Consultant on Optiv's Attack and Penetration team.

Key Accomplishments

James has presented at Black Hat Arsenal and DerbyCon. James has also contributed to the Metasploit Framework and Veil Evasion Toolkit, as well as several other open source security tools.

Nicholas Nam, Research Consulting Director

Nicholas Nam leads and executes highly technical network and application security assessments, as well as adversarial simulation assessments and other advanced research consulting engagements.

Experience

Nick has 19 years of professional experience in the information security space, where the early part of his career was focused on defensive roles at various organizations within the U.S. Department of Defense. Nick has spent the last 8 years within the private sector performing offensive security testing against web applications and networks as well as adversary simulations.

Prior to joining Atredis Partners, Nick was a Security Innovation Principal with FusionX (an Accenture company). While at FusionX, Nick performed many goal-based adversary simulations against some the world’s top communications, financial and media companies.

Key Accomplishments

As part of his work performing adversary simulations, Nick has successfully taken over a national television network’s broadcast system, payroll and wire transfer systems for several large, multinational financial institutions, and gained unrestricted access to secret projects such as proprietary genomic data at a large multinational chemical company.

Holly Leitru, Director of Client Engagement

Holly Leitru leads the execution of client engagement and business development processes at Atredis Partners. She serves as primary liaison between client representatives and various Atredis stakeholders, shepherding engagements through the entire presales flow from scoping initiation to contract execution and scheduling, with the goal of ensuring a smooth and efficient engagement experience for all.

Experience

Holly joined the Operations team at Atredis in 2017, working her way from Client Operations Associate to Client Operations Lead to Director of Client Engagement as of 2024. She previously worked primarily in sports and event management roles responsible for the fulfillment of partnership agreements for professional sports teams and execution of various types of events in a convention center setting. Her particular skills and competencies include plate spinning, juggling calendars, and jumping through procedural hoops.

Key Accomplishments

Holly holds a bachelor's degree in Sport Management with a minor in Communications from Rogers State University, graduating Magna Cum Laude and selected as the Outstanding Graduate from her degree program in 2014. She earned a certificate in Project Leadership from eCornell in 2023.

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 six years, Atredis Partners has doubled in size annually, and has been named three times to the Saint Louis Business Journal's "Fifty Fastest Growing Companies" and "Ten Fastest Growing Tech Companies". Consecutively for the past three years, Atredis Partners has been listed on 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 our client base is by definition confidential and we often operate under strict nondisclosure agreements, Atredis Partners has delivered notable public security research on improving security at Google, Microsoft, The Linux Foundation, Motorola, Samsung and HTC products, and were the first security research firm to be named in Qualcomm's Product Security Hall of Fame. We've received four research grants from the Defense Advanced Research Project Agency (DARPA), participated in research for the CNCF (Cloud Native Computing Foundation) to advance the security of Kubernetes, worked with OSTIF (The Open Source Technology Improvement Fund) and The Linux Foundation on the Core Infrastructure Initiative to improve the security and safety of the Linux Kernel, and have 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 and Advisory 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.